

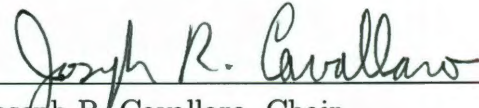
RICE UNIVERSITY
**Parallel VLSI Architectures for Multi-Gbps
MIMO Communication Systems**

by

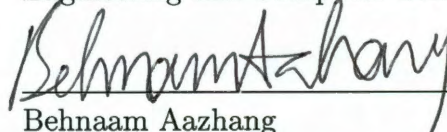
Yang Sun

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE
Doctor of Philosophy

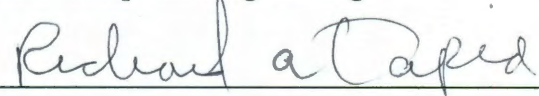
APPROVED, THESIS COMMITTEE:



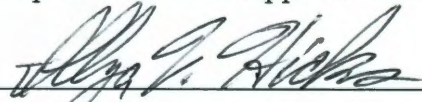
Joseph R. Cavallaro, Chair
Professor of Electrical and Computer
Engineering and Computer Science



Behnaam Aazhang
J.S. Abercrombie Professor of Electrical
and Computer Engineering



Richard A. Tapia
University Professor and
Maxfield-Oshman Professor of
Computational and Applied Mathematics



Illya V. Hicks
Associate Professor of Computational and
Applied Mathematics



Jorma Lilleberg
Adjunct Professor of Electrical and
Computer Engineering

Houston, Texas
December, 2010

ABSTRACT

Parallel VLSI Architectures for Multi-Gbps MIMO Communication Systems

by

Yang Sun

In wireless communications, the use of multiple antennas at both the transmitter and the receiver is a key technology to enable high data rate transmission without additional bandwidth or transmit power. Multiple-input multiple-output (MIMO) schemes are widely used in many wireless standards, allowing higher throughput using spatial multiplexing techniques. MIMO soft detection poses significant challenges to the MIMO receiver design as the detection complexity increases exponentially with the number of antennas. As the next generation wireless system is pushing for multi-Gbps data rate, there is a great need for high-throughput low-complexity soft-output MIMO detector.

The brute-force implementation of the optimal MIMO detection algorithm would consume enormous power and is not feasible for the current technology. We propose a reduced-complexity soft-output MIMO detector architecture based on a trellis-search method. We convert the MIMO detection problem into a shortest path problem. We introduce a path reduction and a path extension algorithm to reduce the search complexity while still maintaining sufficient soft information values for the detection. We avoid the missing counter-hypothesis problem by keeping multiple paths during the trellis search process. The proposed trellis-search algorithm is a data-parallel algorithm and is very suitable for high speed VLSI implementation. Compared with

the conventional tree-search based detectors, the proposed trellis-based detector has a significant improvement in terms of detection throughput and area efficiency. The proposed MIMO detector has great potential to be applied for the next generation Gbps wireless systems by achieving very high throughput and good error performance.

The soft information generated by the MIMO detector will be processed by a channel decoder, e.g. a low-density parity-check (LDPC) decoder or a Turbo decoder, to recover the original information bits. Channel decoder is another very computational-intensive block in a MIMO receiver SoC (system-on-chip). We will present high-performance LDPC decoder architectures and Turbo decoder architectures to achieve 1+ Gbps data rate. Further, a configurable decoder architecture that can be dynamically reconfigured to support both LDPC codes and Turbo codes is developed to support multiple 3G/4G wireless standards.

We will present ASIC and FPGA implementation results of various MIMO detectors, LDPC decoders, and Turbo decoders. We will discuss in details the computational complexity and the throughput performance of these detectors and decoders.

Acknowledgments

I would like to thank my advisor, Professor Joseph R. Cavallaro, for his thoughtful comments and support for the last three years. I would also like to thank other members of my committee, Professor Behnaam Aazhang, Professor Richard Tapia, Professor Illya Hicks, and Professor Jorma Lilleberg for their constructive comments.

I would like to thank Texas Instrument, Xilinx, Nokia, Nokia-Siemens Networks, Synfora/Synopsys, and US National Science Foundation (under grants CCF-0541363, CNS-0551692, CNS-0619767, CNS-0923479, and EECS-0925942) for their support of the research.

I would also like to thank my family. First, to my parents, I could not have accomplished this without your support. Second, to my wife, Qinyi, for being supportive and helpful as always.

Last but not least, I would like to thank Tai Ly, Marjan Karkooti, Predrag Radosavljevic, Kia Amiri, Michael Wu, Guohui Wang, and Bei Yin for their useful feedback and comments.

Contents

Abstract	ii
Acknowledgments	iv
List of Illustrations	x
List of Tables	xv
1 Introduction	1
1.1 Motivation	1
1.2 Scope of The Thesis	5
1.3 Thesis Contribution	5
1.4 Thesis Outline	8
1.5 List of Symbols and Abbreviations	9
2 Background and Related Work	14
2.1 MIMO Detection	14
2.1.1 System Model	14
2.1.2 Maximum Likelihood (ML) Detection	15
2.1.3 Maximum <i>A Posteriori</i> (MAP) Detection	15
2.1.4 Conventional Tree-Search Based MIMO Detection Algorithm .	16
2.2 Error-Correcting Codes	19
2.2.1 Turbo Codes	20
2.2.2 Low-Density Parity-Check Codes	23
2.2.3 Block-structured Quasi-Cyclic (QC) LDPC Codes	27
2.3 Summary and Challenges	29

3	High-Throughput MIMO Detector Architecture	30
3.1	Trellis-Search Algorithm	30
3.1.1	Trellis Graph	31
3.1.2	Multiple Shortest Paths Problem	32
3.1.3	Trellis Traversal Strategies	33
3.1.4	Simulation Result	40
3.1.5	Discussions on Sorting Complexity	46
3.1.6	Discussions on Search Patterns	48
3.2	n-Term-Log-MAP Algorithm	49
3.3	Iterative Detection and Decoding	52
3.4	VLSI Architecture for The Trellis-Search Detector	58
3.4.1	Fully-Parallel Systolic Architecture	58
3.4.2	Path Reduction Unit (PRU)	60
3.4.3	Path Extension Unit (PEU)	66
3.4.4	Path Selection Unit (PSU)	66
3.4.5	LLR Computation Unit (LLRC)	67
3.4.6	Throughput Performance of The Systolic Architecture	69
3.4.7	Folded Architecture	69
3.5	Summary	71
4	High-Throughput Turbo Detector for LTE/LTE-Advanced System	73
4.1	LTE/LTE-Advanced Turbo Codes	75
4.2	QPP Interleaver	76
4.2.1	Algebraic Description of QPP Interleaver	76
4.2.2	QPP Contention-Free Property	78
4.2.3	Hardware Implementation of QPP Interleaver	80
4.3	Sliding Window and Non-Sliding Window MAP Decoder Architecture	83

4.3.1	QPP Interleaving Address Generator for SW-MAP Decoder . . .	87
4.3.2	QPP Address Generator for Radix-4 SW-MAP Decoder	90
4.3.3	QPP Address Generator for NSW-MAP Decoder	94
4.3.4	QPP Address Generator for Radix-4 NSW-MAP Decoder . . .	96
4.3.5	MAP Decoder Comparison	96
4.4	Top Level Parallel Turbo Decoder Architecture	100
4.4.1	Throughput-Area Tradeoff Analysis	104
4.5	Summary	105
5	High-Throughput LDPC Decoder Architecture	109
5.1	Structured QC-LDPC Codes	110
5.2	Layered Decoding Algorithm	112
5.3	Block-Serial Scheduling Algorithm	114
5.4	Min-sum LDPC Decoder Architecture	116
5.4.1	Flexible Permuter Design	119
5.4.2	Pipelined Decoding for Higher Throughput	120
5.5	Log-MAP LDPC Decoder Architecture	121
5.5.1	Low-Complexity Implementation of The Log-MAP Algorithm	121
5.5.2	Radix-2 Log-MAP SISO Decoder	122
5.5.3	Radix-4 SISO Decoder via Look-Ahead Transform	123
5.5.4	Top Level Log-MAP LDPC Decoder Architecture	125
5.5.5	Performance Evaluation	127
5.6	Multi-Layer Parallel LDPC Decoder Architecture	127
5.6.1	Multi-Layer Decoding Performance Evaluation	131
5.6.2	Double-Layer Parallel Decoder Architecture for IEEE 802.11n LDPC Codes	132
5.7	Discussion on the Similarities of LDPC Decoders and Turbo Decoders	139
5.8	Flexible and Configurable LDPC/Turbo Decoder	140

5.8.1	Flex-SISO Module	140
5.8.2	Flex-SISO Module to Decode LDPC Codes	143
5.8.3	Flex-SISO Module to Decode Turbo Codes	145
5.8.4	Design of A Flexible Functional Unit	150
5.8.5	Design of A Flexible SISO Decoder	157
5.8.6	LDPC/Turbo Parallel Decoder Architecture Based on Multiple Flex-SISO Decoders	162
5.9	Summary	163

6 ASIC and FPGA Implementation Results 164

6.1	Decoder Accelerator Design for WARP Testbed	164
6.2	VLSI Implementation Results for MIMO Detectors	169
6.2.1	Trellis-Search MIMO Detector, $M = 1$	169
6.2.2	Trellis-Search MIMO Detector, $M = 2$	170
6.3	VLSI Implementation Results for LTE Turbo Decoders	175
6.3.1	Highly-Parallel LTE-Advanced Turbo Decoder	175
6.4	VLSI Implementation Results for LDPC Decoders	178
6.4.1	IEEE 802.11n LDPC Decoder	178
6.4.2	Variable Block-Size and Multi-Rate LDPC Decoder	179
6.4.3	An IEEE 802.11n/802.16e Multi-Mode LDPC Decoder	181
6.4.4	LDPC Decoder Implementation Using High Level Synthesis Tool	183
6.4.5	Multi-Layer Parallel LDPC Decoder for IEEE 802.11n	186
6.5	VLSI Implementation Results for LDPC/Turbo Multi-Mode Decoder	187
6.5.1	Implementation Results for The Flexible Functional Unit	187
6.5.2	Implementation Results for The Flex-SISO Decoder	188
6.5.3	Implementation Results for The Top-level LDPC/Turbo Decoder	189
6.6	Discussions on the Iterative Receiver Design and Implementation	195
6.7	Summary	197

7 Conclusion and Future Work	199
7.1 Conclusion of The Current Results	199
7.2 Future Work	200
Bibliography	202

Illustrations

1.1	Simplified MIMO system block diagram.	3
2.1	Block diagram for a spatial-multiplexing MIMO system with N_t transmit and N_r receive antennas.	15
2.2	An example tree structure for a MIMO system	18
2.3	Turbo encoder structure	21
2.4	Traditional Turbo decoding procedure using two SISO decoders . . .	22
2.5	Implementation of LDPC decoders	27
2.6	A block structured parity check matrix	28
3.1	A trellis graph for the 4×4 4-QAM system	32
3.2	Flow of the path reduction algorithm	35
3.3	Path reduction example for a 4×4 4-QAM trellis	36
3.4	An example data flow of the path extension algorithm	39
3.5	Path extension example for one node	41
3.6	Frame error rate performance of a coded 4×4 16-QAM MIMO system	43
3.7	Frame error rate performance of a coded 4×4 64-QAM MIMO system	44
3.8	Bit error rate performance of a coded 4×4 16-QAM MIMO system .	45
3.9	Frame error rate performance for one-pass trellis search algorithm . .	49
3.10	Error performance of the n-Term-Log-MAP detection algorithm . . .	53
3.11	Iterative MIMO receiver block diagram	54
3.12	Error performance of an iterative detection and decoding system, $M = 1$	56

3.13 Error performance of an iterative detection and decoding system, $M = 2$	57
3.14 A pipelined fully-parallel “systolic” architecture for the PPTS detector	59
3.15 Block diagram of the PRU	61
3.16 Block diagram of the MFU	62
3.17 Block diagram of the CMP unit	62
3.18 Block diagram of the PCU	65
3.19 Block diagram of the PEDC unit	65
3.20 Block diagram of the PEU	66
3.21 Block diagram of the PSU	67
3.22 Block diagram of the LLRC unit.	68
3.23 Eight-term log-sum unit.	68
3.24 Folded architecture for the PPTS detector.	70
3.25 Detection timing diagram for a 4 antenna system using the folded architecture.	71
4.1 Structure of rate 1/3 Turbo encoder in the LTE/LTE-advanced system.	75
4.2 An example of the contention-free interleaving	79
4.3 Forward QPP address generator circuit diagram, step size = d	82
4.4 Backward QPP address generator circuit diagram, step size = d	83
4.5 Simulation result for a rate of 0.95 LTE Turbo code using two different sliding window algorithms.	86
4.6 Two recommended MAP decoding algorithms for LTE Turbo codes .	88
4.7 SW-MAP decoder architecture.	89
4.8 Interleaver addressing scheme for the SW-MAP decoder	90
4.9 Interleaver for the SW-MAP algorithm	91
4.10 Interleaver for the Radix-4 SW-MAP algorithm	92
4.11 NSW-MAP decoder architecture.	93
4.12 Interleaver for the NSW-MAP algorithm	95

4.13 A hardware architecture for generating interleaving addresses for the Radix-4 NSW-MAP decoder.	96
4.14 Multi-MAP parallel decoding algorithm	98
4.15 Area of a NSW-MAP decoder and a SW-MAP decoder.	99
4.16 AT complexity of a SW-MAP decoder and a NSW-MAP decoder. . .	101
4.17 AT complexity of a Radix-4 SW-MAP decoder and a Radix-4 NSW-MAP decoder.	101
4.18 Parallel decoder architecture	102
4.19 Area-throughput tradeoff analysis for Radix-2 Turbo decoder	106
4.20 Area-throughput tradeoff analysis for Radix-4 Turbo decoder. . . .	107
5.1 Parity check matrix and its factor graph representation	111
5.2 Parity check matrix for block length 1944 bits, code rate 1/2, sub-matrix size $Z = 81$, IEEE 802.11n LDPC code.	112
5.3 Block-serial (BS) scheduling algorithm	116
5.4 Top level min-sum LDPC decoder architecture	117
5.5 Processing Engine (PE)	118
5.6 A 4×4 Barrel shifter network	119
5.7 Pipelined decoding	120
5.8 Radix-2 (R2) SISO decoder architecture	123
5.9 Pipelined decoding schedule	124
5.10 One level look-ahead transform of $f(\cdot)$ recursion	124
5.11 Radix-4 (R4) SISO architecture	125
5.12 Log-MAP LDPC decoder architecture with scalable datapath	126
5.13 Performance comparison of different LUT configurations.	128
5.14 Example of the data conflicts when updating LLRs for two layers. . .	131
5.15 Simulation results for multi-layer parallel decoding algorithm. . . .	133
5.16 Macroblock structure	134

5.17 MB-serial LDPC decoder architecture for the double-layer example. . .	135
5.18 Block diagram for the pipelined Min-sum unit (MSU).	136
5.19 R-Regfile organization.	136
5.20 Pipelined decoding data flow for the double-layer example.	139
5.21 Flex-SISO module.	142
5.22 LDPC decoding using Flex-SISO modules	143
5.23 LDPC decoder architecture based on the Flex-SISO module.	145
5.24 Traditional Turbo decoding procedure using two SISO decoders . . .	146
5.25 Modified Turbo decoding procedure using two Flex-SISO modules . .	147
5.26 Turbo decoder architecture based on the Flex-SISO module.	150
5.27 Turbo ACSA structure	151
5.28 Trellis structure for a single parity check code.	152
5.29 A forward-backward decoding flow to compute the extrinsic LLRs for single parity check code.	153
5.30 MAP processor structure for single parity check code.	154
5.31 Circuit diagram for the LDPC $ f(a, b) $ functional unit.	156
5.32 Circuit diagram for the flexible functional unit (FFU) for LDPC/Turbo decoding.	157
5.33 Flexible SISO decoder architecture.	158
5.34 Data flow graph for Turbo decoding.	160
5.35 Flexible SISO decoder architecture in LDPC mode.	161
5.36 Parallel LDPC/Turbo decoder architecture based on multiple Flex-SISO decoder cores.	163
6.1 WARP testbed, including the custom Xilinx FPGA board and the radio daughtercards.	165
6.2 FEC encoder (verilog black-box) integration with WARP MIMO-OFDM System Generator model.	167

6.3	FEC decoder (verilog black-box) integration with WARP MIMO-OFDM System Generator model.	168
6.4	VLSI layout view of the folded trellis-search MIMO detector ($M = 1$).	170
6.5	VLSI layout view of the systolic trellis-search MIMO detector ($M = 2$).	172
6.6	VLSI layout view of an LTE-advanced Turbo decoder.	178
6.7	VLSI layout view for a variable block-size and multi-rate LDPC decoder.	180
6.8	VLSI layout view of an IEEE 802.11n/802.16e multi-mode LDPC decoder.	182
6.9	Two power reduction techniques	183
6.10	VLSI layout view of the LDPC decoder created from high level synthesis.	184
6.11	Simulation results for a rate 1/2, length 2304 WiMAX LDPC code.	191
6.12	Comparison of the convergence speed.	192
6.13	Simulation results for 3GPP-LTE Turbo codes with a variety of block sizes.	193
6.14	Area estimation for iterative receiver.	196
6.15	Power estimation for iterative receiver.	197

Tables

1.1	Major mobile telecommunication standards.	2
2.1	Commonly used FEC codes in mobile wireless standards.	20
3.1	Sorting complexity comparison	48
4.1	QPP interleaver parallelism.	79
4.2	MAP decoder architecture comparison.	97
5.1	LUT approximation for $g(x) = \log(1 + e^{- x })$	155
5.2	LUT implementation	155
5.3	Functional description of the FFU	158
6.1	Architecture comparison with existing MIMO detectors	171
6.2	Fixed point design parameters for the 4×4 16-QAM MIMO system .	171
6.3	Architecture comparison with two independent works	174
6.4	Architecture comparison with two internal works	175
6.5	Turbo decoder ASIC comparison	177
6.6	IEEE 802.11n LDPC decoder design statistics	179
6.7	Variable-size LDPC decoder comparisons	180
6.8	IEEE 802.11n/802.16e LDPC decoder comparison	181
6.9	LDPC decoder comparisons, HLS v.s. manual design.	185

6.10	SpyGlass power estimates with and without clock gating	185
6.11	Throughput performance of the multi-layer parallel decoder	187
6.12	LDPC decoder comparison for IEEE 802.11n	187
6.13	Synthesis results for different functional units	188
6.14	Flex-SISO decoder area distribution.	189
6.15	Performance of the unified LDPC/Turbo decoder.	190
6.16	Architecture comparison with existing flexible LDPC/Turbo solutions.	195

Chapter 1

Introduction

1.1 Motivation

Mobile wireless connectivity is a key feature of a growing range of devices from laptops and cell phones to digital homes and portable devices. Many applications, such as digital video, are driving the creation of new high data rate multiple antenna wireless algorithms with challenges in the creation of area - time - power efficient architectures.

The mobile telecommunication system has evolved from several Kbps low data-rate 1G (for “first generation”) analog systems to the current 10-100 Mbps enhanced 3G (3.5G, 3.75G, 3.9G) generation. This is soon expected to be followed by 4G with a target data rate of 1 Gbps. Table 1.1 shows a representative set of mobile wireless standards to highlight their differences in data rates.

As an example of the next generation wireless system, 3GPP Long Term Evolution (LTE) [1], which is a set of enhancements to the 3G Universal Mobile Telecommunications System (UMTS) [2], has received tremendous attention recently and is considered to be a very promising 4G wireless technology. For example, Verizon Wireless has decided to deploy LTE in their next generation 4G evolution. One of the main advantages of 3GPP LTE is high throughput. For example, it provides a peak data

Table 1.1 : Major mobile telecommunication standards.

Generation	Technology	Data rates	Year
1G	AMPS, TACS	14.4 Kbps	~1981
2G	GSM, CDMA, TDMA	144 Kbps	~1995
2.5G, 2.75G	GPRS, EDGE, CDMA2000	~200 Kbps	~2000
3G	W-CDMA, CDMA2000 1xEV-DO	384 Kbps	~2002
3.5G, 3.75G, 3.9G	HSDPA, LTE, WiMAX	10-100 Mbps	~2007
4G	IMT-Advanced, LTE-Advanced	1 Gbps	2012+

rate of 172.8 Mbps for a 2×2 antenna system, and a 326.4 Mbps for a 4×4 antenna system for every 20 MHz of spectrum. Furthermore, LTE-Advanced [3], the further evolution of LTE, promises to provide up to 1 Gbps peak data rate.

In order to provide higher data rates, wireless systems are adopting multiple antenna configurations with spatial multiplexing to support parallel streams of wireless data. As an example, the Vertical Bell Laboratories Layered Space-Time (V-BLAST) system has been shown to achieve very high spectral efficiency [4]. There is an increasing demand for Gbps wireless systems. For example, 3GPP LTE-Advanced, IEEE 802.16m WiMAX, IEEE 802.11ac WLAN, and WIGWAM [5] target for Gbps throughput with MIMO technology.

In order to enable reliable delivery of digital data over unreliable wireless channels, the sender encodes the data using an error-correcting code prior to transmission. The additional information (or redundancy) added by the code is used by the receiver to

recover the original data. Error-correcting codes are widely used in MIMO wireless communications. The most commonly used error correcting codes in modern systems are convolutional codes, Turbo codes, and low-density parity-check (LDPC) codes. As a core technology in wireless communications, FEC (forward error correction) coding has migrated from the basic 2G convolutional/block codes to more powerful 3G Turbo codes, and LDPC codes forecast for 4G systems.

Figure 1.1 shows a block diagram of a MIMO system and highlights the Detection and Decoding blocks that are used to recover the multiple transmitted streams. The number of transmit antennas and transmit streams is typically two or four but could be as many as 8 or 12 in future systems. The complexity of the detection and decoding algorithms can vary greatly depending on the number of antennas, modulation, and channel code used in the system.

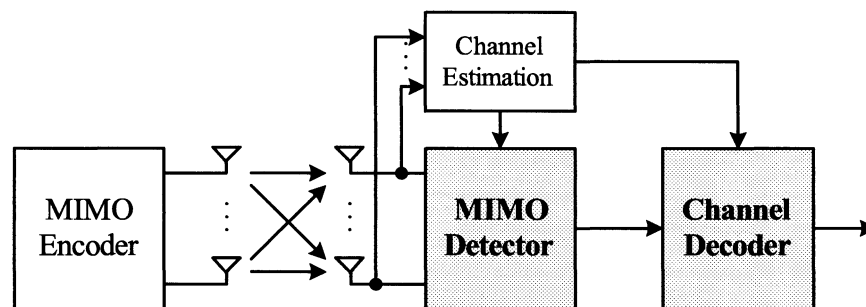


Figure 1.1 : Simplified MIMO system block diagram.

An MIMO detector is used to recover and detect the multiple transmitted streams. Soft-output MIMO detection poses significant challenges to the MIMO receiver design as the computational complexity increases exponentially with the number of antennas.

The optimal soft-decision detector, the maximum *a posteriori* (MAP) detector, will consume enormous computing power and require tremendous computational resources which makes it infeasible to be implemented in a practical MIMO receiver. As such, there is a great need for efficient MIMO algorithms to reduce the MIMO detection complexity.

A channel decoder is used to process the soft information generated by the MIMO detector and reconstruct the original error-free data. Among all those channel decoders, LDPC decoders and Turbo decoders are two of the most important decoders that are widely used in wireless communication systems. Two major challenges of the decoder design are high throughput and flexibility. To support multi-Gbps data rate, we need to develop efficient algorithms and architectures. To support multiple communication standards, we need to develop flexible decoding algorithms and architectures.

As two of the most complex blocks in a wireless receiver, the MIMO detector and the channel decoder consume a significant portion of the silicon area in a wireless receiver SoC (system-on-chip). Thus, it is very important to develop high-throughput low-complexity MIMO detectors and channel decoders to reduce the overall complexity of a wireless SoC.

1.2 Scope of The Thesis

Scope of this thesis is from algorithm to VLSI architecture to ASIC/FPGA implementation. The central part of the thesis is the development of a novel MIMO detection algorithm and architecture, and a flexible LDPC/Turbo decoder architecture. We propose a low-complexity trellis-search algorithm for MIMO detection. We use a trellis graph to represent the search space of the MIMO signal and convert the detection problem into a shortest path problem.

We propose an area-efficient layered decoder architectures for LDPC decoding. We further propose a multi-layer parallel decoding algorithm and architecture for multiple Gbps high throughput decoding of LDPC codes. We propose parallel MAP algorithms for Turbo decoding. By unifying the message passing algorithms of the LDPC codes and the Turbo codes, we develop a configurable LDPC/Turbo architecture.

1.3 Thesis Contribution

This thesis work has generated 20 technical papers, 2 book chapters, and 3 U.S. patent applications.

High-Throughput MIMO Detector [6, 7, 8, 9, 10]: To reduce the MIMO detection complexity, we propose a parallel MIMO detection algorithm and its high-speed VLSI architecture. The proposed detection algorithm is based on a novel path-preserving trellis-search (PPTS) method.

We use a novel trellis graph as an alternative to the tree graph to represent

the search space of the MIMO signal. Based on the trellis graph, we convert the soft MIMO detection problem into a shortest path problem. The proposed PPTS algorithm is a multiple shortest paths algorithm on the condition that every trellis node must be included at least once in this set of paths so that the soft information for every possible symbol transmitted on every antenna is always available. Compared to the traditional tree-search based algorithm, the proposed trellis-search algorithm will have a significantly lower complexity.

The PPTS algorithm is a search-efficient algorithm based on a path-preserving trellis search approach. We introduce a path reduction and a path extension algorithm to reduce the search complexity while still maintaining sufficient soft information values to form the log-likelihood ratios (LLRs) for the transmitted bits. We avoid the missing counter-hypothesis problem by keeping multiple paths during the trellis search process.

The PPTS algorithm is a very data-parallel algorithm because the searching operations at multiple trellis nodes can be performed simultaneously. Moreover, the local search complexity at each trellis node is kept very low to reduce the processing time. Simulation results show that the PPTS algorithm can achieve very good error performance with a low search-complexity. Compared with the conventional tree-search based detectors, the proposed trellis-search detector has a significant improvement in terms of detection throughput and area efficiency. The trellis-search detector has great potential to be applied for the next generation Gbps wireless systems by achiev-

ing very high throughput and good error performance.

Iterative Detection and Decoding: We investigate an iterative detection and decoding algorithm for MIMO communication systems. We modify our trellis-search MIMO detection algorithm to incorporate the *a priori* information from the outer channel decoders, e.g. LDPC decoder and Turbo decoder. Not like the traditional iterative detection and decoding scheme which only performs MIMO detection once, in our scheme, however, we re-run the MIMO detection for each outer iterations to achieve a better performance.

High-Throughput Turbo Decoder [11, 12, 13]: The Turbo decoding algorithm is a sequential algorithm, which makes it very hard to be parallelized. We propose an efficient VLSI architecture for the 3GPP LTE/LTE-Advanced Turbo decoder by utilizing the algebraic-geometric properties of the quadratic permutation polynomial (QPP) interleaver. Turbo interleaver is known to be the main obstacle to the decoder parallelism due to the collisions it introduces in accesses to memory. The QPP interleaver solves the memory contention issues when several MAP decoders are used in parallel to improve Turbo decoding throughput. In this thesis, we propose a low-complexity QPP interleaving address generator and a multi-bank memory architecture to enable parallel Turbo decoding. Design trade-offs in terms of area and throughput efficiency are explored to compare the architectures.

High-Throughput LDPC Decoder [14, 15, 16, 17, 18, 19]: We propose a multi-layer parallel decoding algorithm and VLSI architecture for decoding of struc-

tured quasi-cyclic low-density parity-check (QC-LDPC) codes. The layered decoding algorithm is known to be very memory-efficient and it can achieve a faster convergence speed than the standard two-phase flooding decoding algorithm. In the conventional layered decoding algorithm, the block-rows of the parity check matrix are processed sequentially, or layer after layer. The maximum number of rows that can be simultaneously processed by the conventional layered decoder is limited to the sub-matrix size. To remove this limitation and support layer-level parallelism, we extend the conventional layered decoding algorithm and architecture to enable simultaneous processing of multiple (K) layers of a parity check matrix, which will lead to a K -fold throughput increase. With the proposed decoding algorithm and architecture, a multi-Gbps LDPC decoder is feasible.

ASIC and FPGA Implementation: We have implemented a flexible multi-rate Viterbi decoder for our WARP FPGA testbed. We have also implemented various detectors and decoders on ASICs for throughput, area and power analysis. We have compared the performance of our detectors and decoders against state-of-the-art solutions.

1.4 Thesis Outline

In chapter 2, we will introduce the background of MIMO detection and LDPC and Turbo decoding. We will review the related work in these fields. In chapter 3, we will introduce a trellis-search MIMO detection algorithm and its parallel VLSI

architecture. In chapter 4, we will present a parallel Turbo decoder architecture for LTE/LTE-Advanced system. In chapter 5, we will describe layered LDPC decoding algorithms and architectures for the decoding of the structured QC-LDPC codes. We will further present a flexible LDPC/Turbo joint decoder architecture. In chapter 6, we will summarize the ASIC and FPGA implementation results of various detectors and decoders and compare with existing solutions. Finally, chapter 7 summarizes this thesis.

1.5 List of Symbols and Abbreviations

Here, we provide a summary of the abbreviations and symbols used in this thesis:

ACSA: Add compare select add.

AMPS: Advanced mobile phone system.

APP: *A posteriori* probability.

ASIC: Application-specific integrated circuit.

AWGN: Additive white Gaussian noise.

BICM: Bit interleaved coded modulation.

BPSK: Binary phase shift keying.

CDMA: Code division multiple access.

CDMA2000 1xEV-DO: CDMA evolution-data optimized.

CMP: Comparison.

CMOS: Complementary metal-oxide-semiconductor silicon technology.

dB: Decibel.

DVB-S: Digital Video Broadcasting - satellite.

DVB-T: Digital Video Broadcasting - terrestrial.

EDGE: Enhanced data rates for GSM evolution.

FEC: Forward error correction.

FER: Frame error rate.

FFU: Flexible functional unit.

FPGA: Field-programmable gate array.

Gbps: Gbit/s.

GPRS: General packet radio service.

GSM: Global system for mobile communication.

HDL: Hardware description language.

HLS: High level synthesis.

HSDPA: High-speed downlink packet access.

MAP: Maximum *A Posteriori*.

Mbps: Mbit/s.

MIMO: Multiple-input, multiple-output.

ML: Maximum likelihood.

MFU: Minimum finder unit.

MMSE: Minimum mean square error.

NII: Next iteration initialization.

NSW: Non-sliding window.

LDPC: Low-density parity-check.

LLR: Log-likelihood ratio.

LTE: Long-Term Evolution.

LUT: Look-up table.

OFDM: Orthogonal frequency-division multiplexing.

PCM: Parity check matrix

PE: Processing engines.

PED: Partial Euclidean distance.

PEU: Path extension unit.

PICO: Program-in chip-out.

PPTS: Path-preserving trellis-search.

PRU: Path reduction unit.

PSU: Path selection unit.

QAM: Quadrature amplitude modulation.

QC: Quasi-Cyclic.

QPP: Quadratic permutation polynomial.

RF: Radio frequency.

RTL: Register transfer level.

SISO: Soft-input soft-output.

SMP: State metric propagation.

SNR: Signal-to-noise ratio.

SoC: System-on-chip.

SRAM: Static random access memory.

Sysgen: Xilinx system generator synthesis tool.

TACS: Total access communication system.

TDMA: Time division multiple access

TSMC: Taiwan semiconductor manufacturing company.

UMTS: Universal mobile telecommunications system.

VLSI: Very-large-scale integration.

WCMA: Wideband code division multiple access.

WiMAX: Worldwide interoperability for microwave access.

WLAN: Wireless local area network.

H: Channel matrix in MIMO detection or Parity check matrix in LDPC decoding.

M_c : Number of bits per constellation point.

N_t : Number of transmit antennas.

N_r : Number of receive antennas.

n: Noise vector.

s: Transmitted symbol vector in a MIMO transmitter.

y: Received vector in a MIMO receiver.

H : Superscript denoting the conjugate transpose of a matrix.

T : Superscript denoting the transpose of a matrix.

α : Forward state metrics in Turbo decoding.

β : Backward state metrics in Turbo decoding.

Chapter 2

Background and Related Work

2.1 MIMO Detection

2.1.1 System Model

In this thesis, we consider a spatial-multiplexing MIMO system with N_t transmit antennas and N_r receive antennas ($N_r \geq N_t$), which is shown in Fig. 2.1. The bit-interleaved coded modulation (BICM) is used at the transmitter, where the data bits are multiplexed onto N_t parallel streams. The MIMO transmission can be modeled as a linear system:

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n}, \quad (2.1)$$

where \mathbf{H} is a $N_r \times N_t$ complex matrix and is assumed to be known perfectly at the receiver, $\mathbf{s} = [s_0 \ s_1 \ \dots \ s_{N_t-1}]^T$ is an $N_t \times 1$ transmit symbol vector, \mathbf{y} is an $N_r \times 1$ received vector, and \mathbf{n} is a vector of independent zero-mean complex Gaussian noise entries with variance σ^2 per real component. A real bit-level vector $\mathbf{x}_k = [x_{k,0} \ x_{k,1} \ \dots \ x_{k,B-1}]^T$ is mapped to a complex symbol s_k as $s_k = \text{map}(\mathbf{x}_k)$, where the b -th bit of \mathbf{x}_k is denoted as $x_{k,b}$ and B is the number of bits per constellation point. Through this thesis, symbol s_k and its associated bit vector \mathbf{x}_k will be used interchangeably.

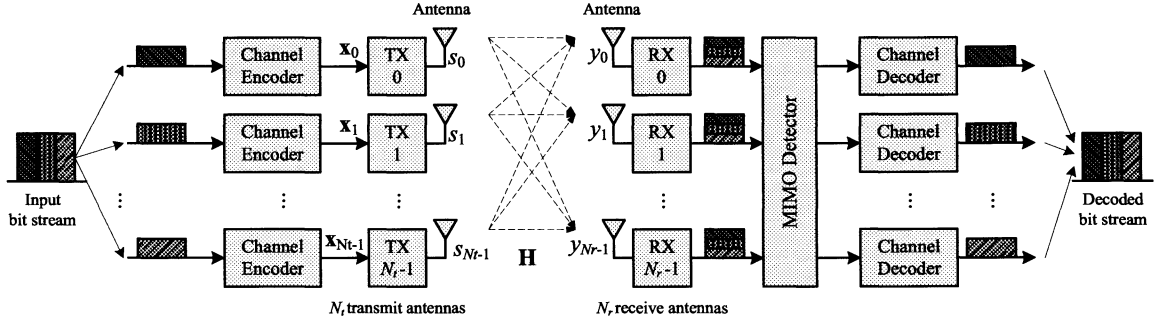


Figure 2.1 : Block diagram for a spatial-multiplexing MIMO system with N_t transmit and N_r receive antennas.

2.1.2 Maximum Likelihood (ML) Detection

The maximum likelihood detector tries to make a hard-decision on the transmitted signal by finding an $\hat{\mathbf{s}}$ which minimizes $\|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2$. ML detection is often used for a MIMO system without an outer error-correcting code, or an un-coded MIMO system.

2.1.3 Maximum *A Posteriori* (MAP) Detection

For a coded MIMO system with an outer error-correcting code, e.g. LDPC code, a soft decision of the transmitted signal is required. The optimal MAP detector is to compute the log-likelihood ratio (LLR) value for the *a posteriori* probability (APP) of each transmitted bit. Assuming there is no *a priori* information for the transmitted bit, the LLR APP of each bit $x_{k,b}$ can be computed as [20]:

$$LLR(x_{k,b}) = \ln \frac{P[x_{k,b} = 0|\mathbf{y}]}{P[x_{k,b} = 1|\mathbf{y}]} = \ln \frac{\sum_{\mathbf{s}: x_{k,b}=0} P(\mathbf{y}|\mathbf{s})}{\sum_{\mathbf{s}: x_{k,b}=1} P(\mathbf{y}|\mathbf{s})} = \ln \frac{\sum_{\mathbf{s}: x_{k,b}=0} \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2\right)}{\sum_{\mathbf{s}: x_{k,b}=1} \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2\right)}. \quad (2.2)$$

With the Max-Log approximation [20], (2.2) is simplified to:

$$LLR(x_{k,b}) \approx \frac{1}{2\sigma^2} \left(\min_{\mathbf{s}: x_{k,b}=1} \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2 - \min_{\mathbf{s}: x_{k,b}=0} \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2 \right). \quad (2.3)$$

Note that to form LLR for bit $x_{k,b}$, both the hypothesis-0 and the hypothesis-1 of bit $x_{k,b}$ are required. Otherwise, the magnitude of the LLR will be undetermined. If a (sorted) QR decomposition of the channel matrix according to $\mathbf{H} = \mathbf{Q}\mathbf{R}$ is used, where \mathbf{Q} and \mathbf{R} refer to a $N_r \times N_t$ unitary matrix and a $N_t \times N_t$ upper triangular matrix, respectively, then (2.3) is changed to:

$$LLR(x_{k,b}) = \frac{1}{2\sigma^2} \left(\min_{\mathbf{s}: x_{k,b}=1} d(\mathbf{s}) - \min_{\mathbf{s}: x_{k,b}=0} d(\mathbf{s}) \right), \quad (2.4)$$

where the Euclidean distance, $d(\mathbf{s})$, is defined as:

$$d(\mathbf{s}) = \|\hat{\mathbf{y}} - \mathbf{R} \cdot \mathbf{s}\|^2 = \sum_{k=0}^{N_t-1} |(\hat{\mathbf{y}})_k - (\mathbf{R}\mathbf{s})_k|^2. \quad (2.5)$$

In the equation above, $\hat{\mathbf{y}} = \mathbf{Q}^H \mathbf{y}$, and $(\cdot)_k$ denotes the k -th element of a vector.

2.1.4 Conventional Tree-Search Based MIMO Detection Algorithm

The MIMO detection problem can be approximately solved using linear algorithms such as zero-forcing detection and minimum mean square error (MMSE) detection.

However, the linear algorithms suffer from significant performance loss compared to the non-linear algorithms. In this thesis, we mainly focus on the non-linear MIMO MAP detection algorithms.

Conventionally, the MIMO detection problem is usually tackled based on tree-search algorithms. The Euclidean distance in (2.5) can be computed backward recursively as $d_k = d_{k+1} + e_k$, where $e_k = \left| \hat{y}_k - \sum_{j=k}^{N_t-1} R_{k,j} s_j \right|^2$. Because of the upper triangular structure of the \mathbf{R} matrix, one can envision this iterative algorithm as a tree traversal problem where each level of the tree represents one k value. Each node has Q children, where Q is the QAM modulation size. Fig. 2.2 shows an example tree-graph. In order to reduce the search complexity, a threshold, C , can be set to discard the nodes with distance $d > C$. Therefore, whenever a node with a $d > C$ is reached, any of its children can be pruned out.

The tree-search algorithms can be often categorized into the depth-first search algorithm and the breadth-first search algorithm. The sphere detection algorithm [21, 22, 23, 24, 25] is a depth-first tree-search algorithm to find the closest lattice point. To provide soft information for outer channel decoders, a modified version of the sphere detection algorithm, or soft sphere detection algorithm, is introduced in [20]. There are many implementations of sphere detectors, such as [26, 27, 28, 29, 30, 31, 32, 33, 34, 35]. However, the sphere detector suffers from non-deterministic complexity and variable-time throughput. The sequential nature of the depth-first tree-search process significantly limits the throughput of the sphere detector especially

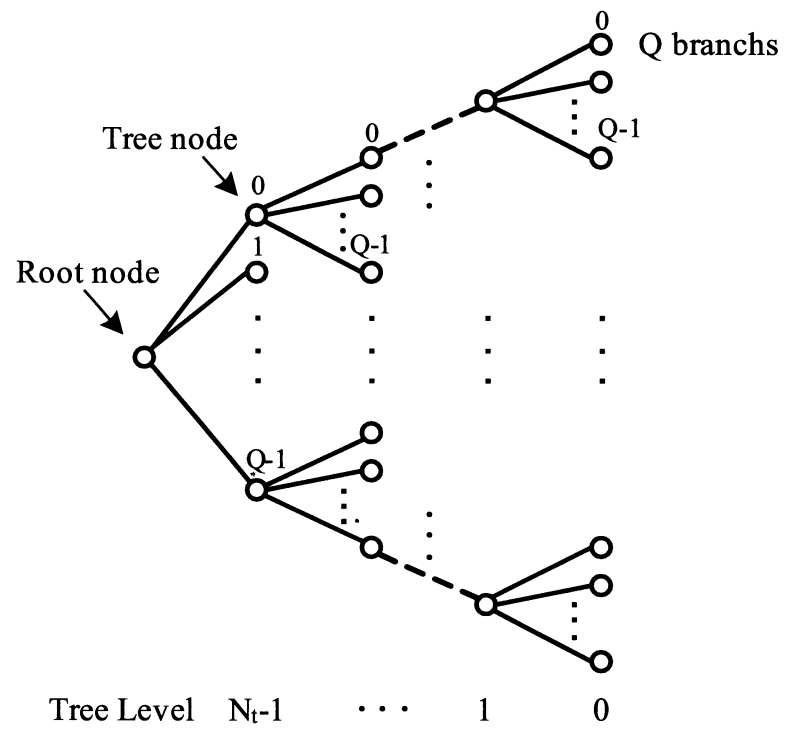


Figure 2.2 : An example tree structure for a MIMO system. The tree has N_t levels. Each tree node has Q children or branches.

when the SNR is low. The K -Best algorithm is a fixed-complexity algorithm based on the breadth-first tree-search algorithm [36, 37, 38, 39, 40, 41]. But this algorithm tends to have a high sorting complexity to find and retain the best candidates, which limits the throughput of the detector especially when K is large. There are some other variations of the K -Best algorithm, which require less sorting than the regular K -best algorithm, e.g. [42, 43, 44, 45, 46], but it is still very difficult for the K -Best detector to achieve 1+ Gbps throughput.

Generally, to make a soft decision for a bit x , a maximum-likelihood (ML) hypothesis and a counter-hypothesis of this bit are both required to form the LLR. A major problem for almost all the “conventional” tree-search algorithms is that the counter-hypotheses for certain bits are missing due to tree pruning. As a consequence of missing counter-hypotheses, the magnitude of the LLRs for certain bits can not be determined, which will lead to performance degradation.

2.2 Error-Correcting Codes

Practical wireless communication channels are inherently “noisy” due to the impairments caused by channel distortions and multipath effects. Error correcting codes are widely used to increase the bandwidth and energy efficiency of wireless communication systems. Table 2.1 summarizes the commonly used forward error correction (FEC) codes in mobile wireless standards. As a core technology in wireless communications, FEC coding has migrated from basic convolutional codes to more powerful Turbo

codes and LDPC codes. Turbo codes, introduced by Berrou *et al.* in 1993 [47], have been employed in 3G and enhanced 3G wireless systems, such as UMTS/WCDMA and 3GPP Long-Term Evolution (LTE) systems. As a candidate for a 4G coding scheme, LDPC codes, which were introduced by Gallager in 1963 [48], have recently received significant attention in coding theory and have been adopted by some advanced wireless systems such as the IEEE 802.16e/802.16m WiMAX system and IEEE 802.11n WLAN system.

Table 2.1 : Commonly used FEC codes in mobile wireless standards.

Generation	Technology	FEC codes
2G	GSM	Convolutional codes
3G	W-CDMA, LTE, WiMAX (802.16e)	Turbo codes
4G	LTE-Advanced, WiMAX (802.16m)	LDPC codes, Turbo codes

2.2.1 Turbo Codes

Turbo codes are a class of high-performance capacity-approaching error-correcting codes [47]. As a break-through in coding theory, Turbo codes are widely used in many 3G/4G wireless standards such as CDMA2000, WCDMA/UMTS, 3GPP LTE, and IEEE 802.16e WiMax.

A classic Turbo encoder structure is depicted in Figure 2.3. The basic encoder consists of two systematic convolutional encoders and an interleaver. The information

sequence \mathbf{u} is encoded into three streams: systematic, parity 1, and parity 2. Here the interleaver is used to permute the information sequence into a second different sequence for encoder 2. The performance of a Turbo code depends critically on the interleaver structure [49].

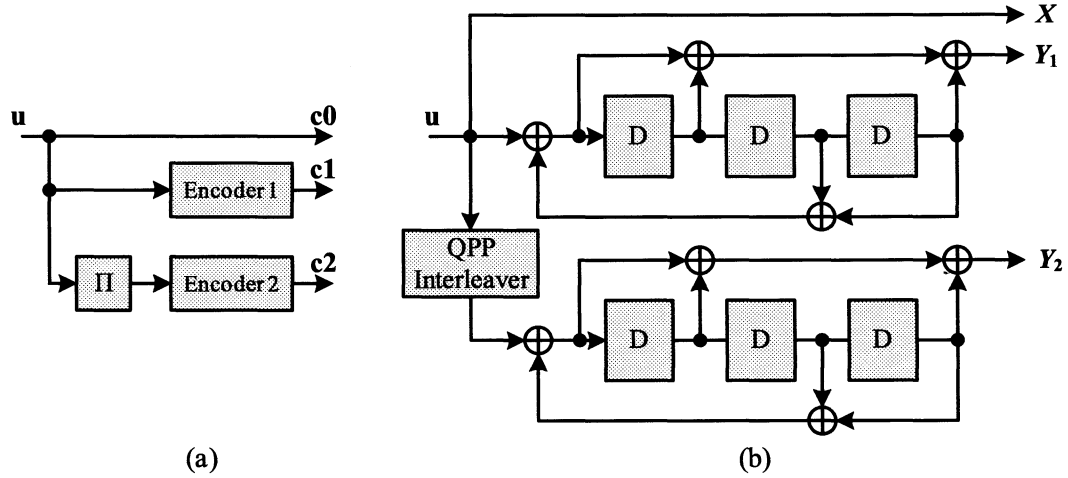


Figure 2.3 : Turbo encoder structure. (a) Basic structure. (b) Structure of Turbo encoder in 3GPP LTE.

The traditional Turbo decoding procedure with two SISO decoders is shown in Fig. 2.4. The definitions of the symbols in the figure are as follows. The information bit and the parity bits at time k are denoted as u_k and $(p_k^{(1)}, p_k^{(2)}, \dots, p_k^{(n)})$, respectively, with $u_k, p_k^{(i)} \in \{0, 1\}$. The channel LLR values for u_k and $p_k^{(i)}$ are denoted as $\lambda_c(u_k)$ and $\lambda_c(p_k^{(i)})$, respectively. The *a priori* LLR, the extrinsic LLR, and the APP LLR for u_k are denoted as $\lambda_a(u_k)$, $\lambda_e(u_k)$, and $\lambda_o(u_k)$, respectively.

In the decoding process, the SISO decoder computes the extrinsic LLR value at

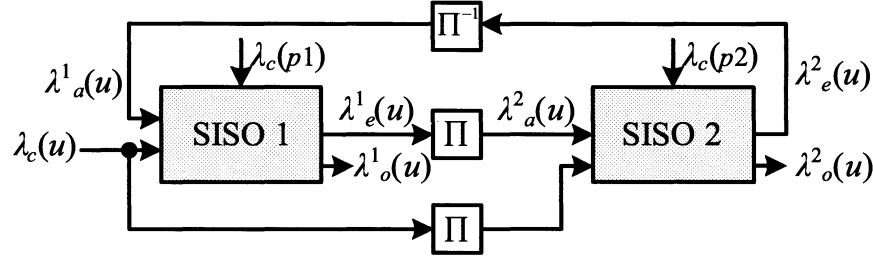


Figure 2.4 : Traditional Turbo decoding procedure using two SISO decoders, where the *extrinsic* LLR values are exchanged between two SISO decoders.

time k as follows:

$$\begin{aligned} \lambda_e(u_k) &= \max_{\mathbf{u}: u_k=1}^* \{ \alpha_{k-1}(s_{k-1}) + \gamma_k^e(s_{k-1}, s_k) + \beta_k(s_k) \} \\ &\quad - \max_{\mathbf{u}: u_k=0}^* \{ \alpha_{k-1}(s_{k-1}) + \gamma_k^e(s_{k-1}, s_k) + \beta_k(s_k) \}. \end{aligned} \quad (2.6)$$

The α and β metrics are computed based on the forward and backward recursions:

$$\alpha_k(s_k) = \max_{s_{k-1}}^* \{ \alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) \} \quad (2.7)$$

$$\beta_k(s_k) = \max_{s_{k+1}}^* \{ \beta_{k+1}(s_{k+1}) + \gamma_k(s_k, s_{k+1}) \}, \quad (2.8)$$

where the branch metric γ_k is computed as:

$$\gamma_k = u_k \cdot (\lambda_c(u_k) + \lambda_a(u_k)) + \sum_i^n p_k^{(i)} \cdot \lambda_c(p_k^{(i)}). \quad (2.9)$$

The *extrinsic* branch metric γ_k^e in (2.6) is computed as:

$$\gamma_k^e = \sum_i^n p_k^{(i)} \cdot \lambda_c(p_k^{(i)}). \quad (2.10)$$

The $\max^*(\cdot)$ function in (2.6-2.8) is defined as:

$$\max^*(a, b) = \max(a, b) + \log(1 + e^{-|a-b|}). \quad (2.11)$$

The soft APP value for u_k is generated as:

$$\lambda_o(u_k) = \lambda_e(u_k) + \lambda_a(u_k) + \lambda_c(u_k). \quad (2.12)$$

In the first half iteration, SISO decoder 1 computes the extrinsic value $\lambda_e^1(u_k)$ and pass it to SISO decoder 2. Thus, the extrinsic value computed by SISO decoder 1 becomes the *a priori* value $\lambda_a^2(u_k)$ for SISO decoder 2 in the second half iteration. The computation is repeated in each iteration. The iterative process is usually terminated after certain number of iterations, when the soft APP value $\lambda_o(u_k)$ converges.

The random interleaver is the main obstacle to the parallel Turbo decoding. To facilitate high speed decoding, new wireless standards are adopting contention-free parallel interleavers. In the literature, many decoder architectures have been extensively investigated for the older 3G Turbo codes [50, 51, 52, 53, 54, 55, 56, 57]. Recently, several Turbo decoders have been developed for the newer 3GPP LTE standard [58, 59, 60, 61]. However, the throughput of those decoders is still below 100 Mbps. As the 4G system standard is pushing for 1 Gbps data rate, it is very important to develop a highly-parallel Turbo decoder architecture.

2.2.2 Low-Density Parity-Check Codes

Low-density parity-check (LDPC) codes [62] have received tremendous attention in the coding community because of their excellent error correction capability and near-capacity performance. Some randomly constructed LDPC codes, measured in bit error rate (BER) performance, come very close to the Shannon limit for the AWGN

channel (within 0.05 dB) with iterative decoding and very long block sizes (on the order of 10^6 to 10^7). The remarkable error correction capabilities of LDPC codes have led to their recent adoption in many standards, such as IEEE 802.11n, IEEE 802.16e, and IEEE 802 10GBase-T.

A binary LDPC code is a linear block code specified by a very sparse binary $M \times N$ parity check matrix:

$$\mathbf{H} \cdot \mathbf{x}^T = 0, \quad (2.13)$$

where \mathbf{x} is a codeword and \mathbf{H} can be viewed as a bipartite graph where each column and row in \mathbf{H} represents a variable node and a check node, respectively. It should be noted the symbol \mathbf{H} used here is different from the symbol \mathbf{H} used for the MIMO channel.

Two-phase Flooding Decoding Algorithm

The basic LDPC decoding algorithm, which is often referred to as the two-phase flooding decoding algorithm, is summarized as follows. We define the following notation. The *a posteriori* probability (APP) log-likelihood ratio (LLR) of each bit n is defined as:

$$L_n = \log \frac{Pr(n = 0)}{Pr(n = 1)}. \quad (2.14)$$

The check node message from check node m to variable node n is denoted as $R_{m,n}$. The variable message from variable node n to check node m is denoted as $Q_{m,n}$. The decoding algorithm is summarized as follows.

Initialization: The variable message $Q_{m,n}$ is initialized to the channel LLR input from the MIMO detection described in Section 2.1.3. The check message $R_{m,n}$ is initialized to 0.

Phase 1) Parity Check Node Update: For each row m , the new check node messages $R'_{m,n}$, corresponding to all variable nodes j that participate in this parity-check equation, are computed using the belief propagation algorithm:

$$R_{m,n} = \prod_{j \in \mathcal{N}_m \setminus n} \text{sign}(Q_{m,j}) \cdot \Psi \left(\sum_{j \in \mathcal{N}_m \setminus n} \Psi(Q_{m,j}) \right), \quad (2.15)$$

where \mathcal{N}_m is the set of variable nodes that are connected to check node m , and $\mathcal{N}_m \setminus n$ is the set \mathcal{N}_m with variable node n excluded. The non-linear function $\Psi(x)$ is defined as:

$$\Psi(x) = -\log \left[\tanh \left(\frac{|x|}{2} \right) \right]. \quad (2.16)$$

To reduce the implementation complexity, the sub-optimal min-sum algorithm [63, 64] can be used to approximate the non-linear function $\Psi(x)$. The scaled min-sum and the offset min-sum algorithms are the two most often used algorithms. For the scaled min-sum algorithm with a scaling factor of S , equation (2.15) is changed to:

$$R_{m,n} \approx S \cdot \prod_{j \in \mathcal{N}_m \setminus n} \text{sign}(Q_{m,j}) \cdot \min_{j \in \mathcal{N}_m \setminus n} |Q_{m,j}|. \quad (2.17)$$

For the offset min-sum algorithm with an offset value of β , equation (2.15) is changed to:

$$R_{m,n} \approx \prod_{j \in \mathcal{N}_m \setminus n} \text{sign}(Q_{m,j}) \cdot \min_{j \in \mathcal{N}_m \setminus n} |Q_{m,j}| - \beta. \quad (2.18)$$

Phase 2) Variable Check Node Update: The APP LLR messages L_n are computed as:

$$L_n = \sum_{j \in \mathcal{M}_n} R_{j,n}, \quad (2.19)$$

where \mathcal{M}_n is the set of check nodes that are connected to variable node n . The variable message is computed as:

$$Q_{m,n} = L_n - R_{m,n}. \quad (2.20)$$

Verification: If all the parity checks are satisfied, the decoding process is finished, otherwise go to phase 1) to start a new iteration.

Hardware Implementation

The hardware implementation of LDPC decoders can be serial, semi-parallel, or fully-parallel. As shown in Fig. 2.5, a fully-parallel implementation has the maximum number of processing elements to achieve very high throughput. A semi-parallel implementation, on the other hand, has a less number of processing elements that can be re-used, e.g. z number of processing elements are employed in Figure 2.5(b). In a semi-parallel implementation, memories are usually required to store the temporary results. In many practical systems, semi-parallel implementations are often employed to achieve several hundred Mbps throughput with reasonable complexity [18, 65, 66, 17, 67, 16, 68].

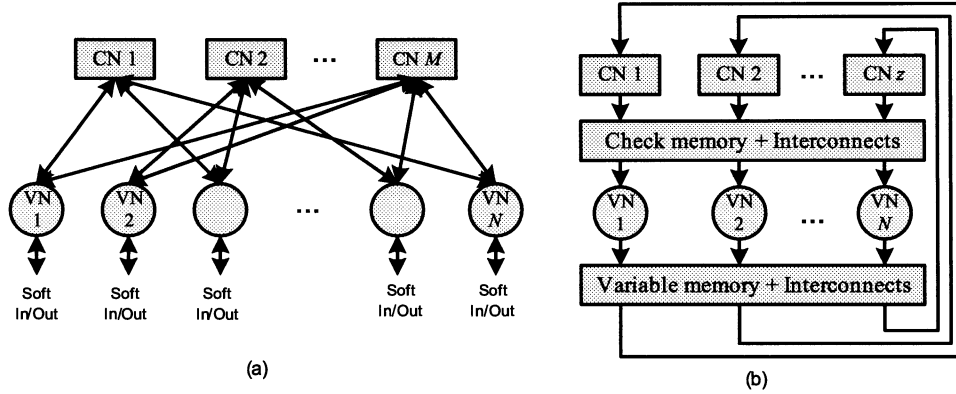


Figure 2.5 : Implementation of LDPC decoders, where CN denotes check node and VN denotes variable node. (a) Fully-parallel. (b) Semi-parallel.

2.2.3 Block-structured Quasi-Cyclic (QC) LDPC Codes

Non-zero elements in \mathbf{H} are typically placed at random positions to achieve good coding performance. However, this randomness is unfavorable for efficient VLSI implementation that calls for structured design. To address this issue, block-structured quasi-cyclic LDPC codes are recently proposed for several new communication standards such as IEEE 802.11n, IEEE 802.16e, DVB-S2 and DMB-T. As shown in Fig. 2.6, the parity check matrix can be viewed as a 2-D array of square sub matrices. Each sub matrix is either a zero matrix or a cyclically shifted identity matrix I_x . Generally, the block-structured parity check matrix \mathbf{H} consists of a $j \times k$ array of $z \times z$ cyclically shifted identity matrices with random shift values x ($0 \leq x < z$). Table 1 summarizes the design parameters for \mathbf{H} in the IEEE 802.11n, IEEE 802.16e, and DMB-T standards.

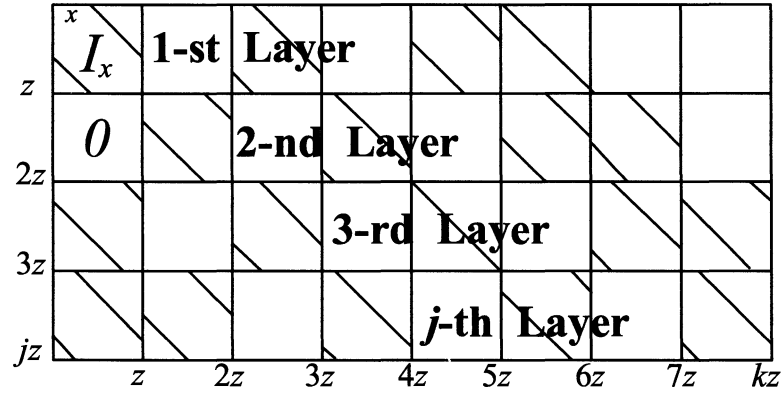


Figure 2.6 : A block structured parity check matrix with block rows (or layers) $j = 4$ and block columns $k = 8$, where the sub-matrix size is $z \times z$.

Table 1: Design parameters for H in several standards

LDPC Code	IEEE 802.11n	IEEE 802.16e	DMB-T
j	4-12	4-12	24-48
k	24	24	60
z	27-81	24-96	127

Flexible LDPC Decoder Architecture

In the recent literature, there are many LDPC decoder architectures [69, 70, 71, 18, 72, 73, 74, 75, 76, 16, 77, 78, 79], but few of them support variable block-size and multi-rate decoding. For example, in [69] a 1 Gbps 1024-bit, rate 1/2 LDPC decoder has been implemented. However this architecture just supports one particular LDPC code by wiring the whole Tanner graph into hardware. In [80], a code rate programmable LDPC decoder is proposed, but the code length is still fixed to 2048 bits for simple VLSI implementation. In [81], a LDPC decoder that supports three block sizes and four code rates is designed by storing 12 different parity check matrices on-chip.

2.3 Summary and Challenges

MIMO detectors and LDPC/Turbo decoders are very complex signal processing blocks in a wireless receiver SoC. The main challenges of the detector and decoder design are high throughput and flexibility. To address these challenges, in chapter 3, we will introduce a low-complexity detection algorithm based on a trellis-search method. We will also present a high-speed VLSI architecture for the trellis-search based MIMO detector. In chapter 4, we will present a high-throughput Turbo decoder for the LTE-Advanced system. In chapter 5, we will describe a multi-mode high-throughput LDPC decoder architecture. In chapter 6, we will assess the hardware implementation tradeoffs for VLSI system design.

Chapter 3

High-Throughput MIMO Detector Architecture

In this chapter, we propose a novel path-preserving trellis-search (PPTS) algorithm and its high-speed VLSI architecture for soft-output MIMO detection. We represent the search space of the MIMO signal with an unconstrained trellis graph. Based on the trellis graph, we convert the soft-output MIMO detection problem into a multiple shortest paths problem subject to the constraint that every trellis node must be covered in this set of paths. The PPTS detector is guaranteed to have soft information for every possible symbol transmitted on every antenna so that the log-likelihood ratio (LLR) for each transmitted data bit can be accurately formed. Simulation results show that the PPTS algorithm can achieve near-optimal error performance with a low search complexity. The PPTS algorithm is a hardware-friendly data-parallel algorithm because the search operations are evenly distributed among multiple trellis nodes for parallel processing.

3.1 Trellis-Search Algorithm

Because the conventional tree-search algorithm is slow and difficult to be parallelized, we propose a search-efficient trellis algorithm to solve the soft MIMO detection problem. The trellis-search algorithm is a data-parallel algorithm that is more suitable

for high-speed hardware implementations.

3.1.1 Trellis Graph

The Euclidean distance in (2.5) can be computed backward recursively. To visualize the recursion, we create a trellis graph. As an example, Fig. 3.1 shows the trellis graph for the 4×4 4-QAM system. In this graph, nodes are ordered into N_t vertical slices or stages, where stage k corresponds to symbol s_k transmitted by antenna k . In other words, the trellis is formed of columns representing the number of transmit antennas and rows representing values of transmitted symbols. The trellis starts with a root node and ends with a dummy sink node. The stages are labeled in descending order. In each stage, there are $Q = 2^B$ different nodes, where each node maps to a constellation point that belongs to a known alphabet. Thus, any transmitted symbol *vector* is a particular *path* through the trellis. The trellis is fully connected, so there are Q^{N_t} number of different paths from root to sink. The nodes in stage k are denoted as $\langle k, q \rangle$, where $q = 0, 1, \dots, Q - 1$. The edge between nodes $\langle k, q \rangle$ and $\langle k - 1, q' \rangle$ has a weight of $e_{k-1}(\mathbf{q}^{(k-1)})$:

$$e_{k-1}(\mathbf{q}^{(k-1)}) = \left| \hat{y}_{k-1} - \sum_{j=k-1}^{N_t-1} R_{k-1,j} \cdot s_j \right|^2, \quad (3.1)$$

where $\mathbf{q}^{(k-1)}$ is the partial symbol vector $\mathbf{q}^{(k-1)} = [q_{k-1} \ q_k \ \dots \ q_{N_t-1}]^T$, and s_j is the complex-valued symbol $s_j = \text{map}(q_j)$. We define the path weight as the sum of the edge weights along this path. Then the weight of a path from root to sink is an Euclidean distance $\|\hat{\mathbf{y}} - \mathbf{R} \cdot \mathbf{s}\|^2$. Define a (partial) path metric d_k as the sum of the

edge weights along this (partial) path. Then the path weight is computed backward recursively as:

$$d_{k-1}(q') = d_k(q) + e_{k-1}(\mathbf{q}^{(k-1)}), \quad (3.2)$$

where $d_{N_T}(\cdot)$ is initialized to 0, and $d_0(\cdot)$ is the path weight (or Euclidean distance).

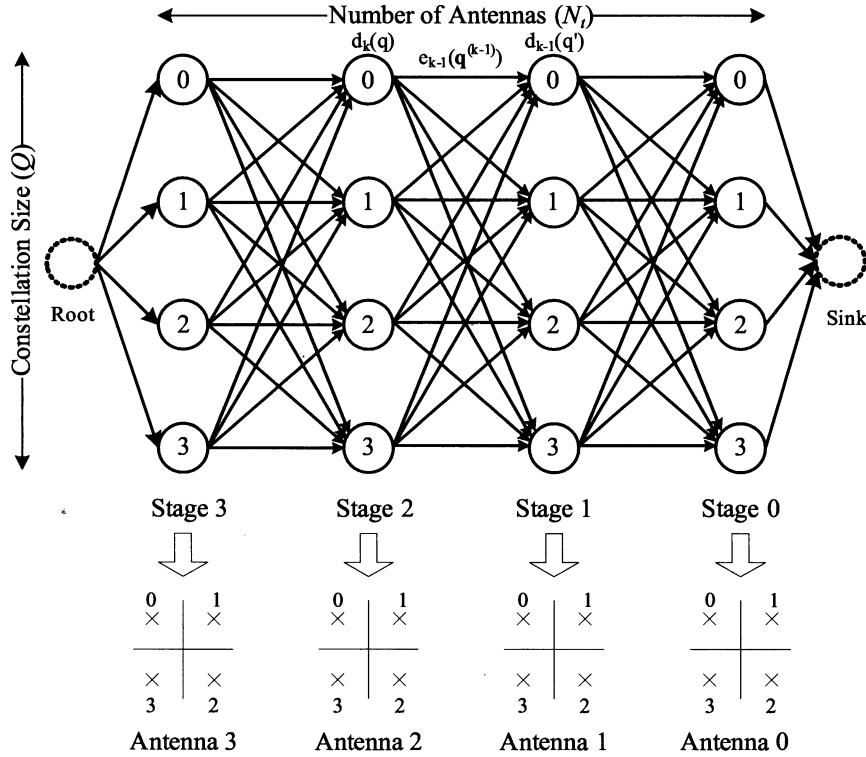


Figure 3.1 : A trellis graph for the 4×4 4-QAM system. Each stage of the trellis corresponds to a transmit antenna. There are $Q = 2^B$ nodes in each stage, where each node maps to a constellation point that belongs to a known alphabet.

3.1.2 Multiple Shortest Paths Problem

We transform the soft MIMO detection problem into a multiple shortest paths problem. A similar technique of shortest path to cover different states in a state space has

been investigated in the graph theory application [82]. In this thesis, we apply the shortest path algorithm to the MIMO detection problem.

In the trellis graph, each trellis node $\langle k, q \rangle$ maps to a complex symbol s_k such that any path from root to sink maps to a particular symbol vector \mathbf{s} . A path weight is a measurement of the soft probability ($P(\mathbf{y}|\mathbf{s})$) for nodes (symbols) on this path. To make a soft decision for every transmitted bit $x_{k,b}$, finding one shortest path is not enough. We want to find multiple paths which cover every node in the trellis graph. The multiple shortest paths problem is defined as follows. *For each node $\langle k, q \rangle$ in the trellis graph, find a shortest path from root to sink that must include this node $\langle k, q \rangle$.* The corresponding shortest path weight is related to the symbol probability ($P(\mathbf{y}|s_k)$). If we can find such a conditional shortest path for each node in the trellis, we will then have one soft information value for every possible symbol transmitted on every antenna. As a result, we will have sufficient soft information values to avoid the missing counter-hypothesis problem. Thus, the LLR for every data bit can be formed accurately based on these soft information values.

3.1.3 Trellis Traversal Strategies

Because of the unconstrained trellis structure, there are Q^{N_t} different paths from root to sink that need to be evaluated. In order to reduce the search complexity, we propose a greedy algorithm that approximately solves the multiple shortest paths problem defined above. In this search algorithm, the trellis is pruned by removing the

unlikely paths. However, we always preserve a predefined number of paths at each trellis node so that there is enough soft information to compute LLRs. We refer to it as the path-preserving trellis-search (PPTS) algorithm. It is a two-step algorithm which is summarized as follows.

Step 1: Path Reduction

The path reduction algorithm is used to prune the unlikely paths in the trellis by applying the M -algorithm [83] locally at each node. Fig. 3.2 illustrates the basic data flow of the path reduction algorithm. Note that Fig. 3.2 illustrates only three successive stages, k , $k - 1$, and $k - 2$ among the N_t stages. Each node receives QM incoming path candidates from nodes in the previous stage of the trellis and, then, and the (M) paths are preserved from these QM candidates. Next, the number M survivors are fully extended to the right so that each node will have the best QM outgoing paths forwarded to the next stage of the trellis.

We define the following notation to help explain the algorithm. Let $\beta_k^{(m)}(j, i)$ denote the QM incoming path candidates for node $\langle k, i \rangle$, and $\alpha_k^{(m)}(i)$ denote the M surviving path metrics selected by node $\langle k, i \rangle$. In Fig. 3.2, the stages of the trellis are labeled in descending order, starting from $N_t - 1$ and ending with 0. In stage k , each node $\langle k, i \rangle$ evaluates its QM incoming path candidates $\beta_k^{(m)}(j, i)$ and selects the best M paths from $\beta_k^{(m)}(j, i)$, where the m -th best path metric is $\alpha_k^{(m)}(i)$. The α metrics are sorted so that $\alpha_k^{(0)}(i) < \alpha_k^{(1)}(i) < \dots < \alpha_k^{(M-1)}(i)$. Next, each of the

surviving paths is fully extended for the next stage so that there are QM outgoing paths leaving from each node $\langle k, i \rangle$, which are $\beta_{k-1}^{(m)}(i, j)$. This search process repeats for every stage of the trellis. The details of the path reduction algorithm are summarized in Algorithm 1.

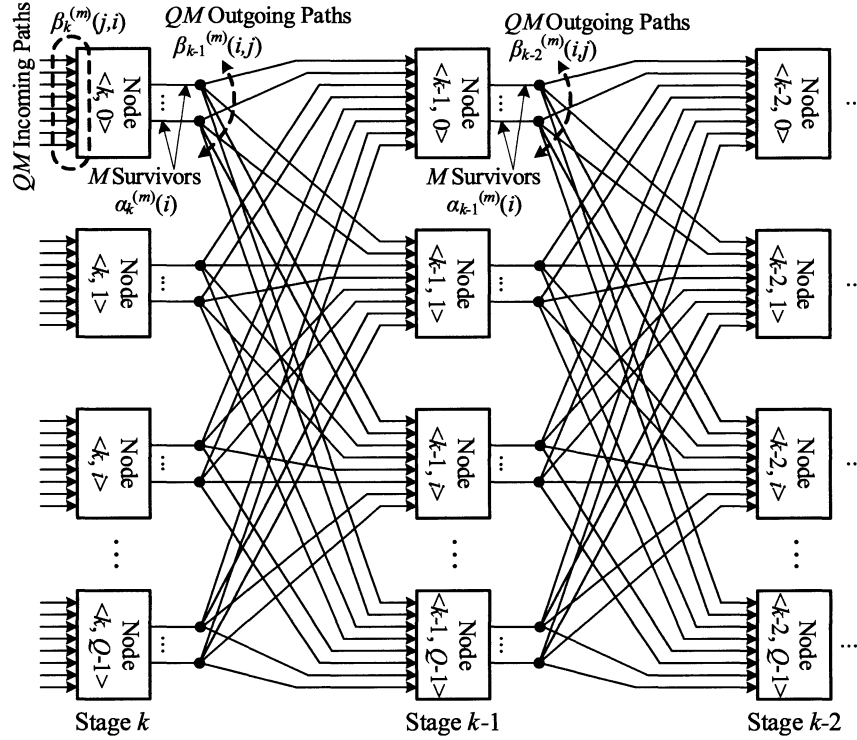


Figure 3.2 : Flow of the path reduction algorithm, where each node evaluates all its incoming paths and selects the best M paths.

As an example, Figure 3.3 shows 4×4 4-QAM trellis graph after applying the path reduction procedure, where each node preserves only $M = 2$ best incoming paths, the one with the least cumulative path weights. The path reduction procedure can effectively prune the trellis by keeping only the number M of the best incoming paths at each trellis node. As a result, each node in the last stage, i.e. stage 0, has the

Algorithm 1 Path Reduction Algorithm

0) **Initialization:** Set loop variable $k = N_t - 1$. For each node $\langle k, i \rangle$, initialize

$$\beta_k^{(m)}(j, i) = \begin{cases} |\hat{y}_k - R_{k,k}s_k(i)|^2, & j, m = 0. \\ +\infty, & j, m \neq 0. \end{cases}$$

1) **Main Loop:**

1.a) *Path Selection:* For each node $\langle k, i \rangle$, select the best M paths $\alpha_k^{(m)}(i)$ from the QM path candidates $\beta_k^{(m)}(j, i)$.

1.b) *Path Calculation:*

for $(0 \leq i \leq Q - 1)$

for $(0 \leq m \leq M - 1)$

for $(0 \leq j \leq Q - 1)$

$$\beta_{k-1}^{(m)}(i, j) = \alpha_k^{(m)}(i) + e_{k-1}^{(m)}(\mathbf{j}^{(k-1)}),$$

where $e_{k-1}^{(m)}(\mathbf{j}^{(k-1)})$ is the edge weight as defined in (3.1).

1.c) *Loop Update:* Set $k = k - 1$. If $k \neq 0$, goto 1.a).

2) **Final Selection:** For each node $\langle 0, i \rangle$, select the best M paths $\alpha_0^{(m)}(i)$ from the QM path candidates $\beta_0^{(m)}(j, i)$.

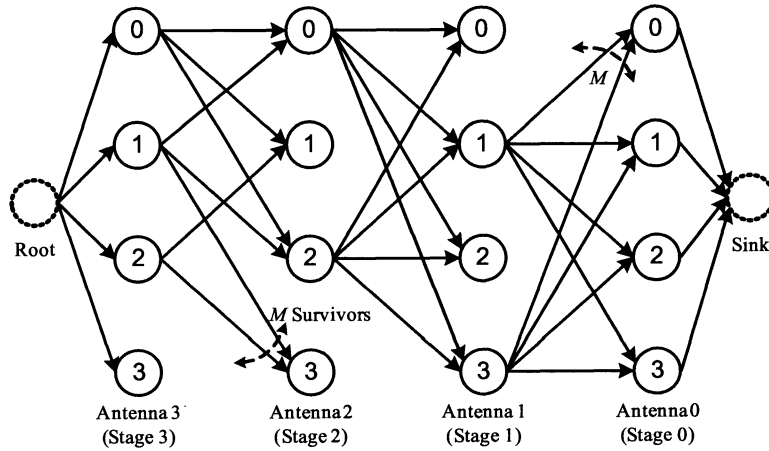


Figure 3.3 : Path reduction example for a 4×4 4-QAM trellis, where $M = 2$ incoming paths are preserved at each node.

number M shortest paths $(\alpha_0^{(m)}(i))$ through the trellis. Recall that each trellis node in stage k maps to a possible symbol s_k in a constellation. Thus, we have obtained a soft information value for every possible symbol s_0 , the symbol transmitted by antenna 0. This is sufficient to guarantee that both the ML hypothesis and the counter-hypothesis in the Max-Log LLR calculation of (2.4) are available for every data bit $x_{0,b}$ transmitted by antenna 0. Then, the LLRs for data bits $x_{0,b}$, $b = 0, 1, \dots, \log Q - 1$, can be computed as:

$$LLR(x_{0,b}) = \frac{1}{2\sigma^2} \left(\min_{i:b=-1} \alpha_k^{(m)}(i) - \min_{i:b=+1} \alpha_k^{(m)}(i) \right), \text{ where } k, m = 0. \quad (3.3)$$

However, other than the trellis nodes in the last stage, the algorithm can not guarantee that every trellis node will have the number M shortest paths through the trellis. For example, in Figure 3.3, nodes $\langle 2, 1 \rangle$ and $\langle 2, 3 \rangle$ have only uncompleted paths. Thus, we may not have enough soft information values to calculate the LLRs for data bits $x_{k,b}$ transmitted by antenna $k \neq 0$ because the counter-hypotheses for these bits can be missing. Although we can use LLR clipping [20] to saturate the LLR values, there will be some performance loss. To preserve enough soft information values for each data bit, we next introduce a path extension algorithm to fill in the missing paths for each trellis node q in stage k .

Step 2: Path Extension

To obtain soft information for every possible symbol s_k , we need to make sure every node in stage k is included in a path from root to sink. To extend node $\langle k, i \rangle$,

we start to travel the trellis from this node and try to find the M most likely paths from this node to the sink node. This is achieved by extending the paths stage by stage, where the best M extended paths are selected in every stage. Fig. 3.4 shows an example data flow for the path extension for one node $\langle k, i \rangle$. Note that instead of waiting for the entire path reduction operation to finish, we will start the path extension operation for antenna k as soon as the path reduction algorithm has finished processing stage k of the trellis. In Fig. 3.4 for example, to detect antenna k , we first perform path reduction from stage $N_t - 1$ to stage k , and next we perform path extension from stage t ($t = k - 1$) to stage 0. Note that only one node's path extension process is shown in this figure. In fact, we will extend all the nodes in stage k simultaneously.

We define the following notation to help explain the algorithm. Let $\theta^{(m)}(k, i, t, j)$ denote the QM extended path candidates from node $\langle k, i \rangle$ to nodes $\langle t, j \rangle$, where $j = 0, 1, \dots, Q - 1$ and $m = 0, 1, \dots, M - 1$. Let $\gamma^{(m)}(k, i, t)$ denote the M surviving paths selected in stage t , where $m = 0, 1, \dots, M - 1$. To extend node $\langle k, i \rangle$, we first retrieve data $\beta_{k-1}^{(m)}(i, j)$ computed in the path reduction algorithm, and use it to initialize $\theta^{(m)}(k, i, t, j) = \beta_{k-1}^{(m)}(i, j)$, where $t = k - 1$. Next, the best M extended paths $\gamma^{(m)}(k, i, t)$ are selected from $\theta^{(m)}(k, i, t, j)$. Then, $\gamma^{(m)}(k, i, t)$ are fully extended for the next stage to form $\theta^{(m)}(k, i, t - 1, j)$. Again, the best M extended paths $\gamma^{(m)}(k, i, t - 1)$ are selected from $\theta^{(m)}(k, i, t - 1, j)$. This process repeats. Finally, $\gamma^{(m)}(k, i, 0)$ are the result M extended paths from node $\langle k, i \rangle$ to the sink node.

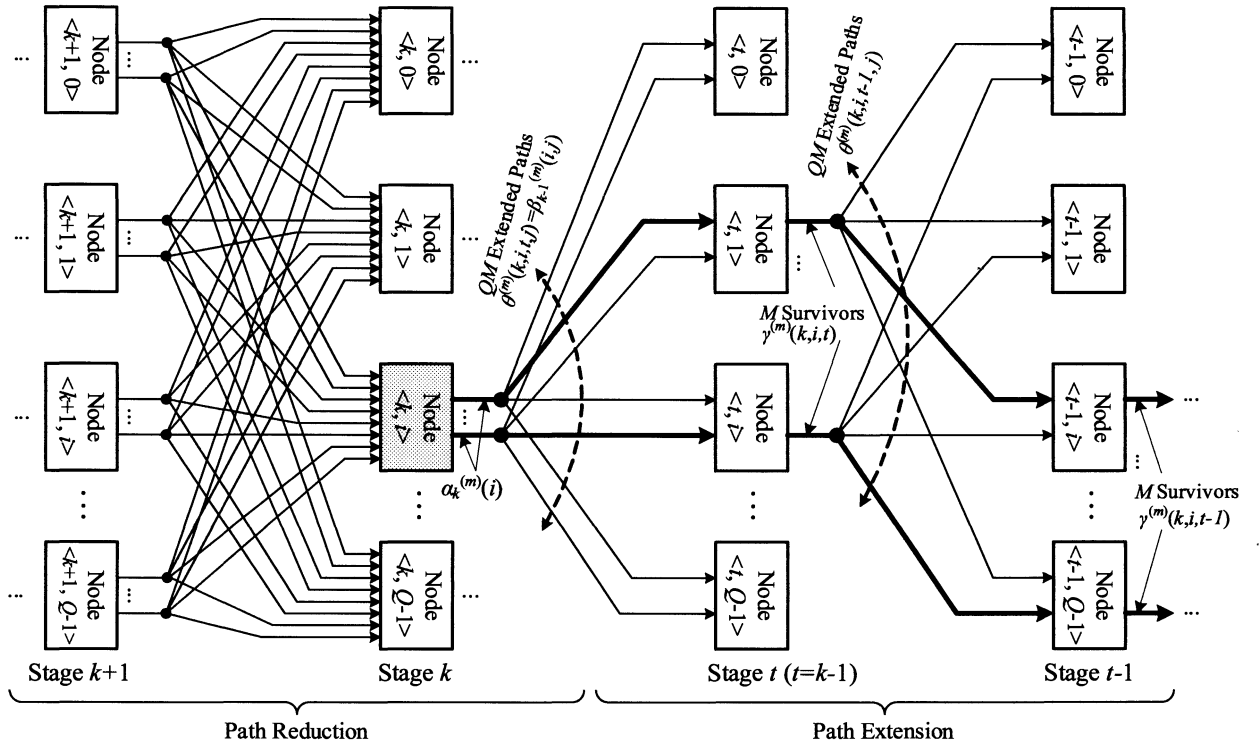


Figure 3.4 : An example data flow of the path extension algorithm for extending one node $\langle k, i \rangle$, where M paths are extended from this node to each of the following stages $(t, t-1, \dots, 0$, where $t = k-1$). All the nodes $\langle k, i \rangle$, $i = 0, 1, \dots, Q-1$, can be extended in parallel.

The path extension algorithm is summarized in Algorithm 2.

Fig. 3.5 shows an example to extend node $\langle 2, 1 \rangle$ in a 4×4 4-QAM trellis. We can see that $M = 2$ paths are extended from this node to the sink node. It should be noted that nodes $\langle k, 0 \rangle, \langle k, 1 \rangle, \dots, \langle k, Q - 1 \rangle$ can be extended in parallel since there is no data dependency between them. After the path extension is finished, every node in stage k will be included in a path from root to sink. Thus, we have obtained a soft information value for every possible symbol s_k , the symbol transmitted by antenna k . This is sufficient to guarantee that both the ML hypothesis and the counter-hypothesis are available for every data bit $x_{k,b}$. Then, the LLRs for data bits transmitted by antenna $k \neq 0$ can be computed as:

$$LLR(x_{k,b}) = \frac{1}{2\sigma^2} \left(\min_{i:b=-1} \gamma^{(m)}(k, i, t) - \min_{i:b=+1} \gamma^{(m)}(k, i, t) \right), \text{ where } t, m = 0. \quad (3.4)$$

Note that although we keep M paths for each node $\langle k, i \rangle$ in every extension step, we only use the final smallest path weight for each node, i.e. $\gamma^{(m=0)}(k, i, t = 0)$, in (3.4) to compute the LLR. However, keeping multiple paths in the intermediate steps helps to improve the accuracy of the LLR values.

3.1.4 Simulation Result

In this section, we evaluate the error performance of the proposed PPTS detector through computer simulations. The floating-point simulations are carried out for 4×4 16-QAM and 4×4 64-QAM systems where the channel matrices are assumed to have independent random Gaussian distributions. A sorted QR decomposition

Algorithm 2 Path Extension Algorithm for Antenna k , $k = N_t - 1, N_t - 2, \dots, 1$

0) **Initialization:** Set loop variable $t = k - 1$. For each node $\langle k, i \rangle$, initialize $\theta^{(m)}(k, i, t, j) = \beta_{k-1}^{(m)}(i, j)$.

1) **Main Loop:**

1.a) *Path Selection:* For each node $\langle k, i \rangle$, select the best M paths $\gamma^{(m)}(k, i, t)$ from the QM path candidates $\theta^{(m)}(k, i, t, j)$.

1.b) *Path Calculation:*

for ($0 \leq i \leq Q - 1$)

for ($0 \leq m \leq M - 1$)

for ($0 \leq j \leq Q - 1$)

$$\theta^{(m)}(k, i, t - 1, j) = \gamma^{(m)}(k, i, t) + e_{t-1}^{(m)}(\mathbf{j}^{(t-1)}),$$

where $e_{t-1}^{(m)}(\mathbf{j}^{(t-1)})$ is the edge weight as defined in (3.1).

1.c) *Loop Update:* Set $t = t - 1$. If $t \neq 0$ goto 1.a).

2) **Final Selection:** For each node $\langle k, i \rangle$, select the best M paths $\gamma^{(m)}(k, i, 0)$ from the QM path candidates $\theta^{(m)}(k, i, 0, j)$.

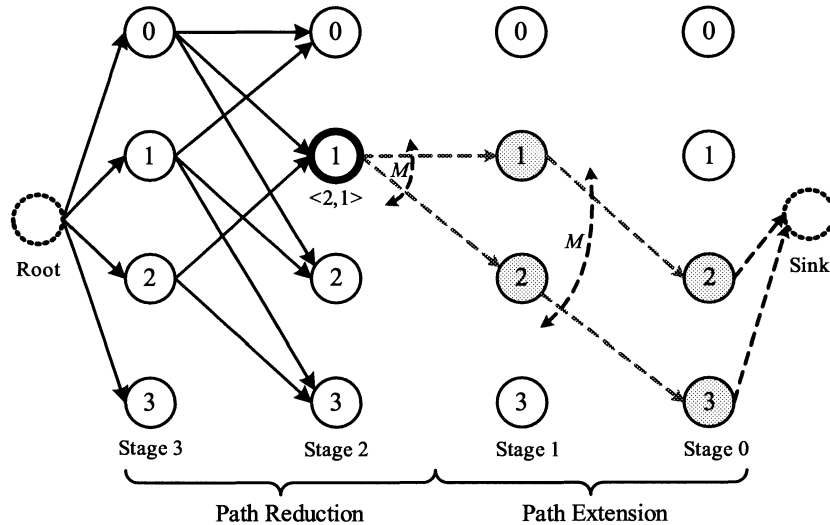


Figure 3.5 : Path extension example for one node $\langle 2, 1 \rangle$, where $M = 2$ paths are extended from this node to the sink node.

of the channel matrix is used. The soft-output of the detector is fed to a length 2304, rate 1/2 WiMax layered LDPC decoder, which performs up to 20 LDPC inner iterations. Figures 3.6 and 3.7 show the frame error rate (FER) performance of the PPTS detectors for different M values. As a reference, we also show the error performance of a Max-Log MAP detector with exhaustive search criterion, and a soft K -Best detector with $K = 4Q$. In the error performance comparison, the Max-Log MAP detector with full search criterion is considered as the baseline reference. We also show a bit error rate (BER) performance for the 4×4 16-QAM system in Figure 3.8.

For a 4×4 16-QAM system, when $M = 1$, the PPTS detector shows about 1 dB performance loss at FER 10^{-3} compared to the baseline reference. When $M = 2$, the PPTS detector shows about 0.35 dB performance degradation. When $M = 3$, the PPTS detector shows only 0.15 dB performance degradation. When $M = 4$, the PPTS detector achieves a performance almost the same as the baseline reference. Compared to the K -Best detector with $K = 32$, the PPTS detectors with $M = 2, 3, 4$ significantly outperform the K -Best detector.

For a 4×4 64-QAM system, when $M = 1$, the PPTS detector shows about 0.75 dB performance loss at FER 10^{-3} compared to the baseline reference. When $M = 2$, the PPTS detector shows about 0.3 dB performance degradation. When $M = 3, 4$, the PPTS detector achieves a performance that is very close to the baseline reference. Compared to the K -Best detector with $K = 64$, the PPTS detector with $M = 1$

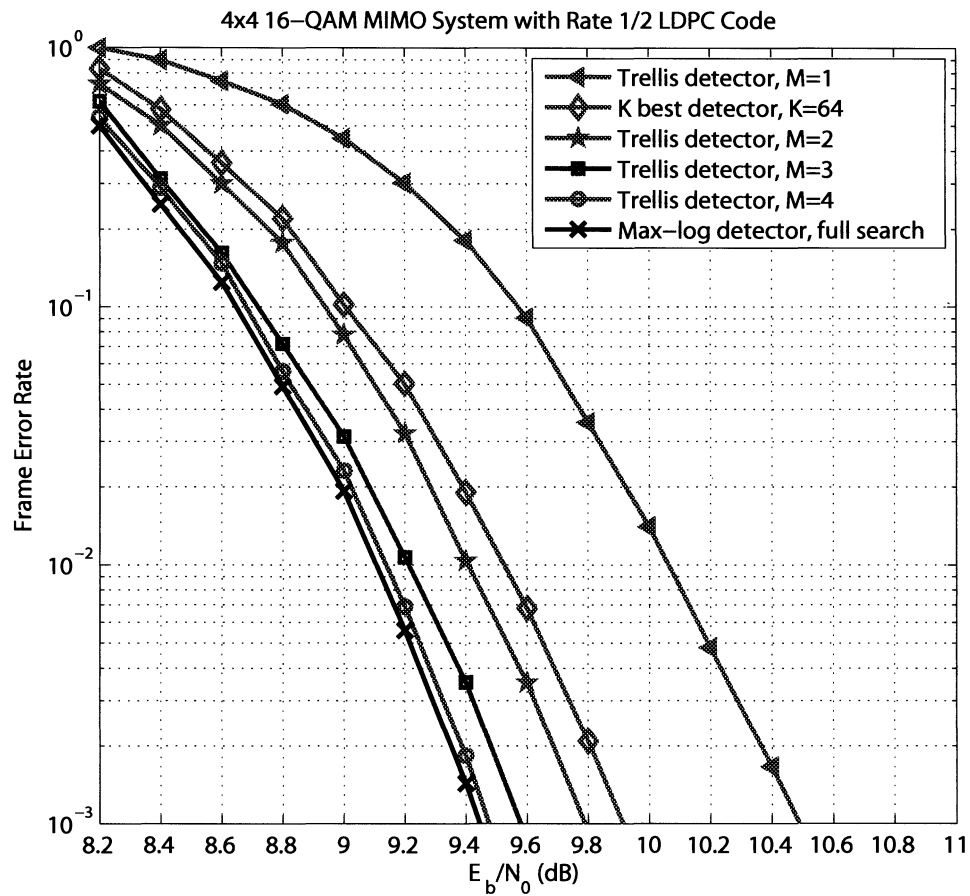


Figure 3.6 : Frame error rate performance of a coded 4×4 16-QAM MIMO system using the PPTS detection algorithm with different M values.

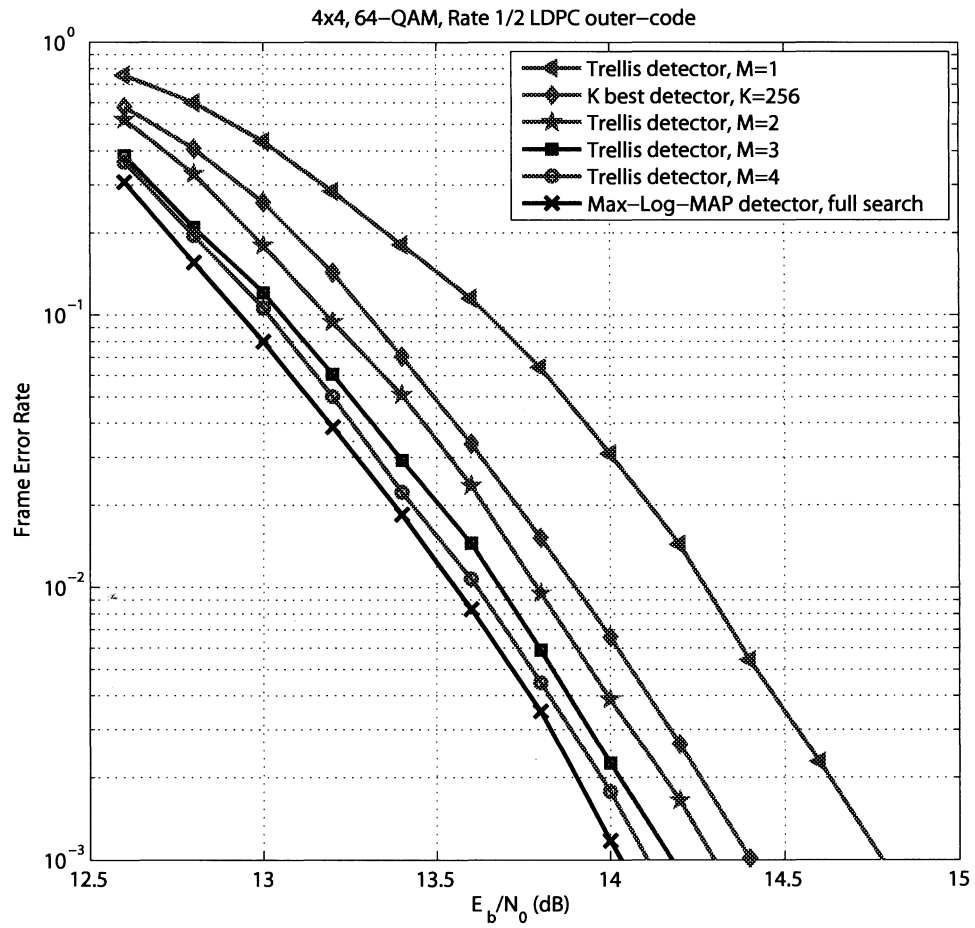


Figure 3.7 : Frame error rate performance of a coded 4×4 64-QAM MIMO system using the PPTS detection algorithm with different M values.

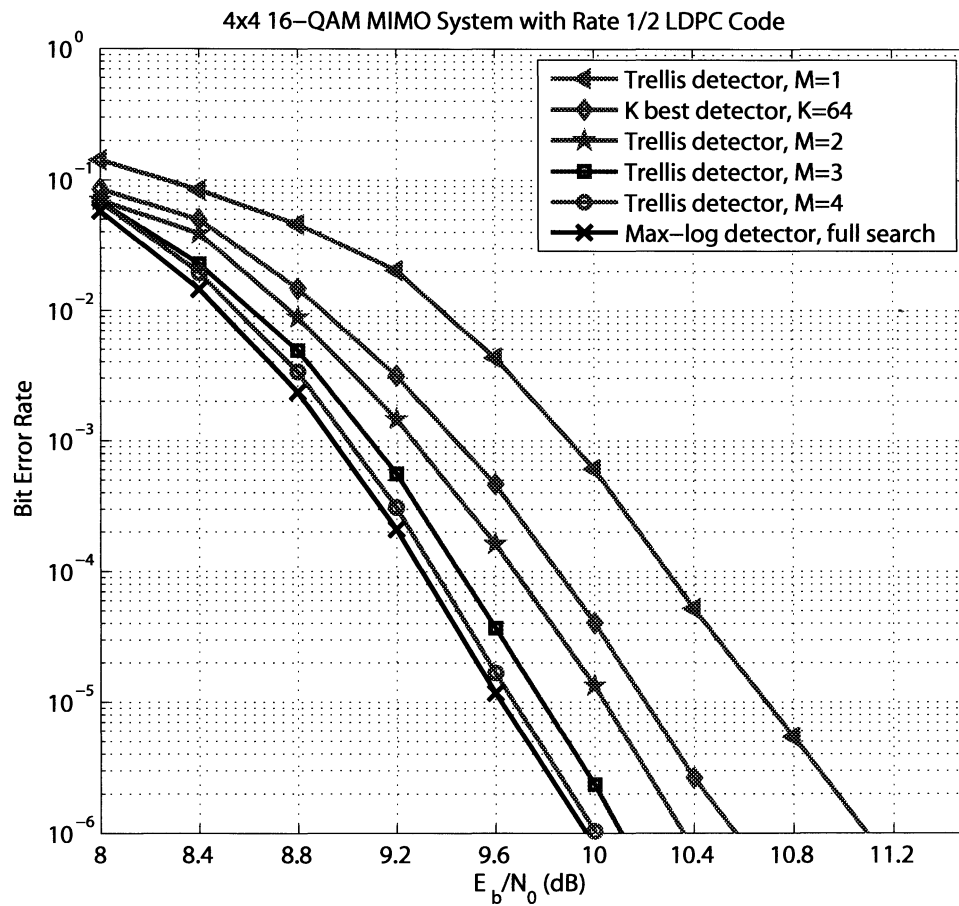


Figure 3.8 : Bit error rate performance of a coded 4×4 16-QAM MIMO system using the PPTS detection algorithm with different M values.

performs similarly to the K -Best detector, but the PPTS detectors with $M = 2, 3, 4$ significantly outperform the K -Best detector.

3.1.5 Discussions on Sorting Complexity

The trellis-search algorithm is a variation of the K -best tree-search algorithm. In the K -best tree-search algorithm, K global candidates are selected in each level of the tree. One limitation of the K -Best tree-search algorithm is that it may not preserve enough soft information for every transmitted bit x . Thus the missing counter-hypothesis problem may occur, which will lead to significant performance loss. On the other hand, the trellis-search algorithm always guarantees that for each transmitted bit x , there will be a ML-hypothesis and a counter-hypothesis so that the LLR for transmitted bit x can be more reliably formed.

Sorting is often the bottleneck in the K -best detectors. Now we compare the sorting cost of the proposed PPTS detector with that of the K -best detector. Both PPTS and K -best detectors need to carry out a (s, t) sorting operation: find the smallest s values out of t candidates. From the above simulation results, we know that the error performance of the K -best detector with $K = 4Q$ is worse than the proposed PPTS detector with $M = 2$. To have a fair comparison, we compare the (s, t) sorting complexity of the more complex PPTS detector with $M = 2$ and the K -best detector with $K = 4Q$.

Table 3.1 summarizes the sorting complexity comparisons. The sorting complex-

ity is measured by the number of pairwise comparisons. Generally, to find the s smallest values from t candidates requires at least $t - s + \sum_{t+1-s < j \leq t} \lceil \log j \rceil$ pairwise comparisons [84]. This bound is only achievable for $s = 1, 2$. For the PPTS detector, Q concurrent (M, QM) sorting operations are required at each trellis stage. For the K -best detector, one global (K, QK) sorting operation is required at each tree level. The (s, t) sorting complexity of the K -best algorithm is approximated by $4(t - 1) + (s - 1) \log_2 t$ when applying the typically used heap sort algorithm [38]. From Table 3.1, we can see that the PPTS detector has a significantly lower sorting complexity than the traditional K -best detector especially for the higher modulation systems. In addition, the PPTS detector can employ Q concurrent smaller sorters which will lead to a significant processing speedup.

The PPTS detector compares favorably than the sort-free detectors, such as the flex-sphere detector [85] and the SSFE detector [44]. These sort-free detectors use a simpler algorithm to avoid the expensive sorting operations at a cost of some performance degradation. It should be noted that even the sort-free detectors avoid the sorting, they still can not achieve more than 300 Mbps throughput for the 4×4 16-QAM system. On the other hand, our trellis-based detector uses a sort-light algorithm to achieve near-optimal performance and multi-Gbps throughput.

Table 3.1 : Sorting complexity comparison

4×4 16-QAM MIMO System		
	K -Best, $K = 64$	Trellis, $M = 2$
Sorting complexity per tree level/trellis stage	$(64, 1024) \sim 4722$ One global sorter	$(2, 32) = 35$ 16 sorters in parallel
Processing speedup	1	135 times faster
Required SNR for 10^{-3} FER	10.0 dB	9.9 dB
4×4 64-QAM MIMO System		
	K -Best, $K = 256$	Trellis, $M = 2$
Sorting complexity per tree level/trellis stage	$(256, 16384) = 69102$ One global sorter	$(2, 128) = 133$ 64 sorters in parallel
Processing speedup	1	520 times faster
Required SNR for 10^{-3} FER	14.4 dB	14.3 dB

3.1.6 Discussions on Search Patterns

In the proposed trellis-search algorithm, we need to perform a multi-pass search operations. In the first-pass, the trellis is pruned by only keeping the best M incoming paths at each node. Next, the trellis is re-visited to fill in the uncompleted paths. One variation of this algorithm is to only visit the trellis once by keeping both M incoming paths and M outgoing paths at each node during the sweep. This algorithm reduces the search complexity at a cost of some performance loss because the edge weight changes as the path changes. Fig. 3.9 compares the frame error performance of the one-pass trellis-search detector with that of the multi-pass trellis-search detector. As can be seen, the one-pass trellis-search has a performance loss of 0.4 dB. However,

the one-pass detector can save the computational operations by about 40%. Thus, the one-pass detector is a tradeoff between complexity and performance.

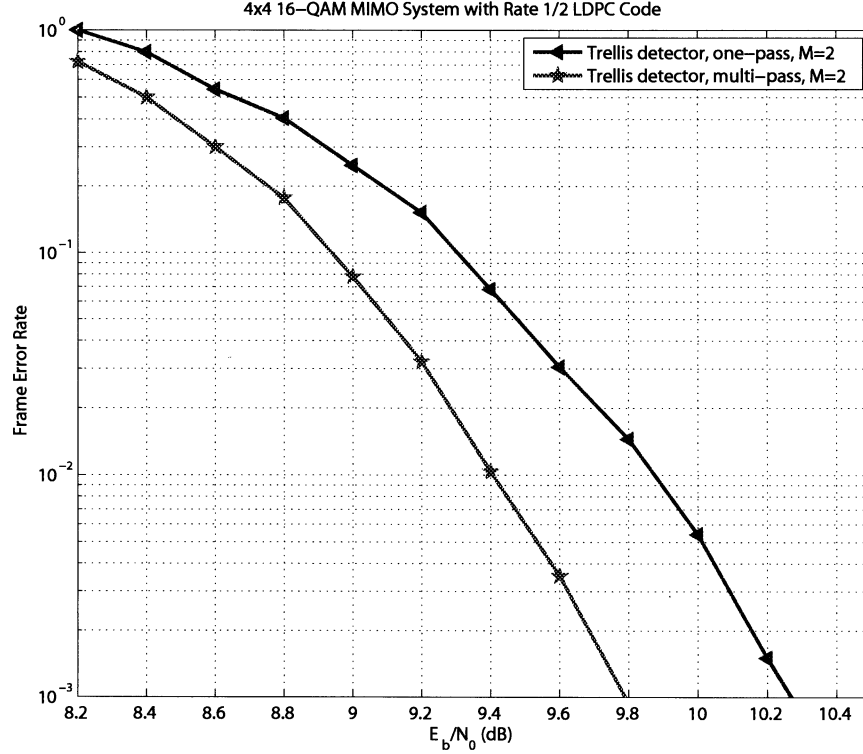


Figure 3.9 : Frame error rate performance for one-pass trellis search algorithm.

3.2 n-Term-Log-MAP Algorithm

As an enhancement to the conventional Max-Log-MAP algorithm, we describe a n-Term-Log-MAP approximation algorithm to achieve near-optimum MIMO detection performance. The same trellis-search algorithm can be used to implement the n-Term-Log-MAP approximation algorithm.

As we know, the optimum soft MIMO detection is based on the Log-MAP algorithm, which is too complex to be implemented in a practical MIMO receiver because the Log-MAP algorithm requires calculating log-sum of $\frac{QM}{2}$ exponential terms, where Q is the constellation size and M is the number of transmit antennas. In practice, the Log-MAP algorithm is often approximated by the Max-Log-MAP algorithm to reduce complexity. However, there is still a performance gap between the sub-optimum Max-Log-MAP detector and the optimal Log-MAP detector. Almost all the existing MIMO detector implementations are based on the sub-optimal Max-Log-MAP approximation which limits the error performance of the detector.

In this section, we propose a reduced-complexity Log-MAP approximation algorithm for high performance MIMO detection. In the proposed algorithm, we use a reduced number (n) of exponential terms to approximate the original Log-MAP algorithm as:

$$LLR(x_{k,b}) = \ln \sum_{i=0:x_{k,b}=0}^{n-1} \exp \left(-\frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2 \right) - \quad (3.5)$$

$$\ln \sum_{i=0:x_{k,b}=1}^{n-1} \exp \left(-\frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2 \right). \quad (3.6)$$

The trellis search method described before can be modified to implement the n-Term-Log-MAP algorithm. Recall that in the trellis search algorithm, each node keeps a list of M most likely paths. So altogether QM candidates in each stage k of the trellis can be used to compute the LLRs for data bits transmitted by antenna k using the n-Term-Log-MAP algorithm, where $n = \frac{QM}{2}$.

The n-term log-sum operation can be implemented by iteratively applying the two-term log-sum. The two-term log-sum can be computed using the advantageous Jacobean algorithm as follows:

$$\ln(e^a + e^b) = \max(a, b) + \ln(1 + e^{|a-b|}) = \max^*(a, b). \quad (3.7)$$

The $\ln(1+e^{|a-b|})$ can be approximated by using a one-dimension look-up table accessed by $|a - b|$. Then the n-term log-sum can be recursively computed using the Jacobean algorithm. The following equation shows an example to implement a four-term log-sum:

$$\max^*(a, b, c, d) = \max^*(\max^*(a, b), \max^*(c, d)). \quad (3.8)$$

To further reduce the complexity, we break the computation into two steps. Recall that each stage of the trellis corresponds to a transmit antenna, and each node in a stage is mapped to a constellation point. We can first compute a symbol reliability metric $\Gamma(q)$ for each node q as follows

$$\Gamma_k(q) = \ln \sum_{l=0}^{L-1} e^{-\frac{1}{2\sigma^2} d_k^{(l)}(q)} = \max_l \left(-\frac{1}{2\sigma^2} d_k^{(l)} \right). \quad (3.9)$$

The LLR for each transmitted bit is computed as:

$$LLR(x_{k,b}) = \max_{q: x_{k,b}=0} \left(-\frac{1}{2\sigma^2} d_k^{(l)} \right) - \max_{q: x_{k,b}=1} \left(-\frac{1}{2\sigma^2} d_k^{(l)} \right). \quad (3.10)$$

Since multiple exponential terms are used, this algorithm will significantly outperform the Max-Log-MAP algorithm. Given a modulation size Q , the local list size M determines the decoding performance: larger M value leads to better error performance.

It should be noted that the n-Term-Log-MAP algorithm can not be applied to the traditional MIMO detection algorithms such as the K-best detector and the sphere detector because they can not guarantee that multiple exponential terms will exist when computing LLRs. This is because in the tree search process, the tree nodes are not grouped by their QAM values. Therefore, there is no control of how many terms are found for each possible constellation point.

We evaluate the error performance of the proposed n-Term-Log-MAP trellis-search detector. The floating-point simulations are carried out for a 4x4 16-QAM system where the channel matrices are assumed to have independent random Gaussian distributions. A (2304, 1152) WiMax LDPC code is used as an outer channel code. As references, we also plot the simulation results for the optimal Log-MAP detector, the Max-Log-MAP detector based on the exhaustive search, and the Max-Log-MAP detector based on the K-Best search algorithm. As can be seen from Fig. 3.10, the n-Term-Log-MAP detector with $M = 2$ significantly outperforms the K-Best detector with $K = 32$. The n-Term-Log-MAP detector with $M = 3$ outperforms the Max-Log-MAP detector with exhaustive search criterion. The n-Term-Log-MAP detector with $M = 4$ and $M = 6$ performs very close to the optimal Log-MAP algorithm.

3.3 Iterative Detection and Decoding

Iterative detection and decoding is a technique to combine the detection and decoding process to further improve the performance. By exchanging information between the

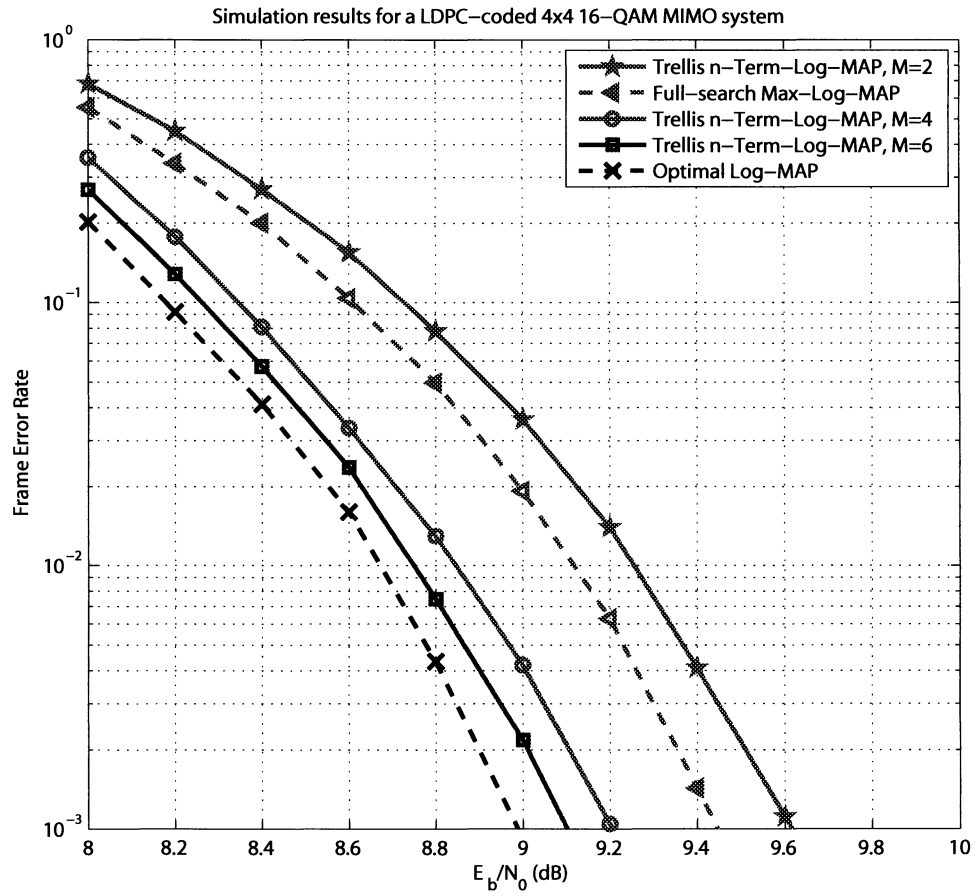


Figure 3.10 : Error performance of a coded 4×4 16-QAM MIMO system using the n-Term-Log-MAP detection algorithm with different M values.

detector and the decoder, an iterative receiver has a significant performance improvement over the non-iterative receiver.

In an iterative detection and decoding scheme [20] as illustrated in Fig. 3.11, the MIMO detector generates extrinsic information L_{E1} using the received signal \mathbf{y} and the *a priori* information L_{A1} provided by the channel decoder. In the first iteration, L_{A1} is not available and is assumed to be 0.

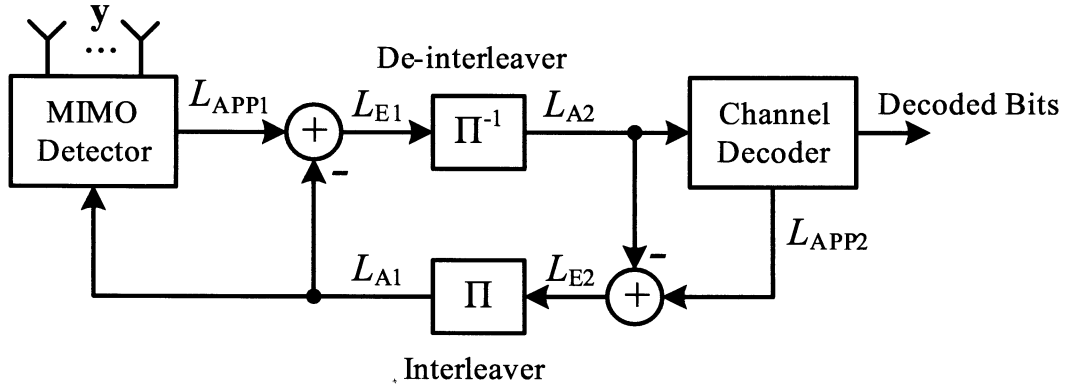


Figure 3.11 : Iterative MIMO receiver block diagram, where the subscript “1” denotes soft information associated with the MIMO detector and the subscript “2” denotes soft information associated with the channel decoder.

Now the LLR value for each bit $x_{k,b}$ is changed to: [20]

$$LLR(x_{k,b}) = \ln \frac{\sum_{\mathbf{s}: x_{k,b}=0} \exp \left(-\frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2 + \sum_{k=0}^{N_t-1} \sum_{b=0}^{B-1} x_{k,b} \cdot L_A(x_{k,b}) \right)}{\sum_{\mathbf{s}: x_{k,b}=1} \exp \left(-\frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2 + \sum_{k=0}^{N_t-1} \sum_{b=0}^{B-1} x_{k,b} \cdot L_A(x_{k,b}) \right)}. \quad (3.11)$$

where $L_A(x_{k,b})$ is the *a priori* LLR value for bit $x(k,b)$. With the Max-Log approximation, the LLR value of (3.11) is simplified to

$$LLR(x_{k,b}) = \frac{1}{2\sigma^2} \left(\min_{\mathbf{s}: x_{k,b}=1} d(\mathbf{s}) - \min_{\mathbf{s}: x_{k,b}=0} d(\mathbf{s}) \right), \quad (3.12)$$

where the Euclidean distance, $d(\mathbf{s})$, is defined as:

$$d(\mathbf{s}) = \sum_{k=0}^{N_t-1} \left(|(\hat{\mathbf{y}})_k - (\mathbf{R}\mathbf{s})_k|^2 - \sigma^2 \sum_{b=0}^{B-1} x_{k,b} \cdot L_A(x_{k,b}) \right). \quad (3.13)$$

In a traditional iterative MIMO receiver implementation [86, 87], because the detection block is often the bottleneck, the detection is performed only once. A list of candidates generated by the MIMO detector are stored in a list buffer. In each outer iteration, the soft values generated by the channel decoder are only fed back to the list buffer to update the list and generate new soft values based on the new list. A major drawback of this scheme is that the error performance is not as good as the original iteration detection and decoding scheme as shown in Fig. 3.11.

However, with the proposed trellis-search algorithm, the MIMO detection task can be performed very fast. Therefore, it is realistic to re-run the entire detection in each outer iteration. The same trellis-search algorithm can be used for the iterative MIMO detector by modifying the original edge weight function (3.1) to:

$$e_{k-1}(\mathbf{q}^{(k-1)}) = \left| \hat{y}_{k-1} - \sum_{j=k-1}^{N_T-1} R_{k-1,j} \cdot s_j \right|^2 - \sigma^2 \sum_{j=k-1}^{N_t-1} \sum_{b=0}^{B-1} x_{j,b} \cdot L_A(x_{j,b}). \quad (3.14)$$

The error performance of the iterative detection and decoding scheme is evaluated through computer simulations. The floating-point simulations are carried out for 4×4 16-QAM systems where the channel matrices are assumed to have independent random Gaussian distributions. A (2304, 1152) WiMax LDPC code is used as an outer channel code. The outer LDPC iteration is fixed to 20. The magnitude of the extrinsic LLR L_{E1} is saturated to 15 to avoid the large LLR values with a wrong

sign. Fig. 3.12 shows the error performance of the iterative receiver based on the $M = 1$ trellis-search max-log-MAP detector for different outer iterations. Fig. 3.13 shows the error performance of the iterative receiver based on the $M = 2$ trellis-search max-log-MAP detector for different outer iterations. As can be seen, with one outer iteration, the FER performance can be improved by 1.5 to 2 dB. By increasing the number of the outer iterations, the FER performance can be increased by about 2.5 to 3 dB.

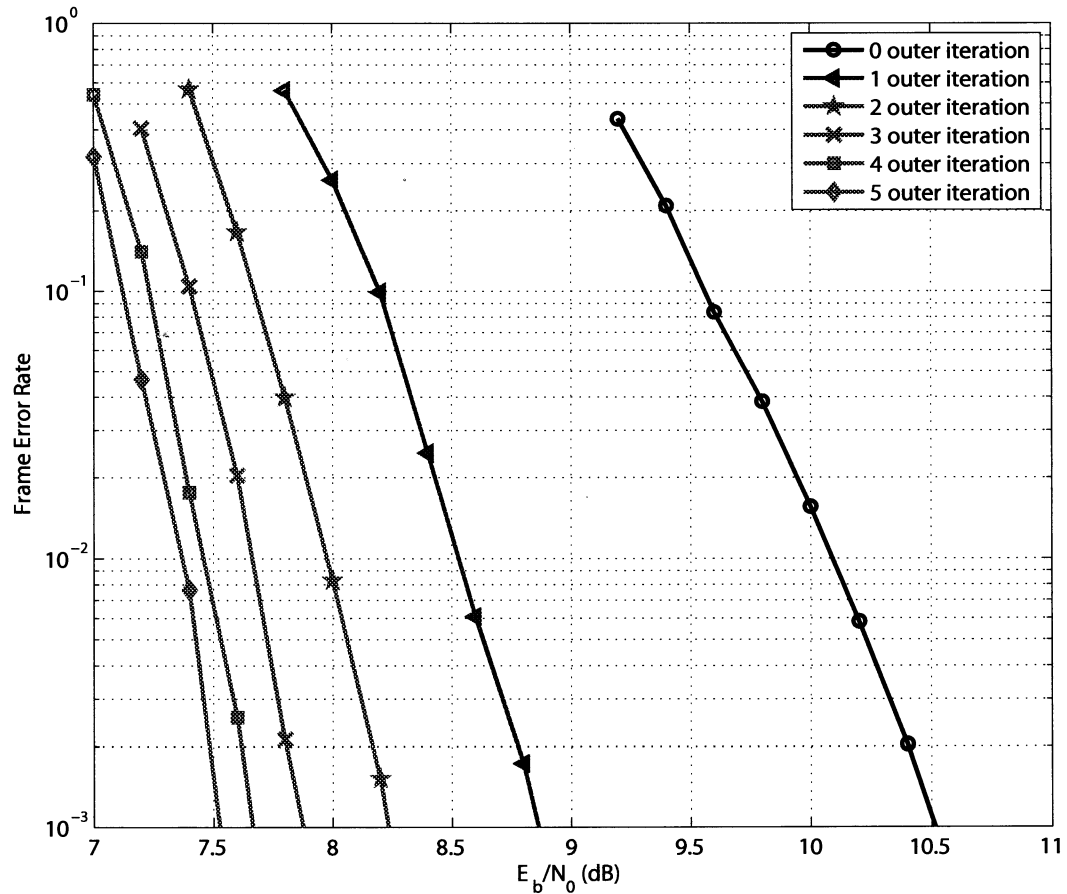


Figure 3.12 : Error performance of an iterative detection and decoding system, where a $M = 1$ trellis-search max-log-MAP detector is used.

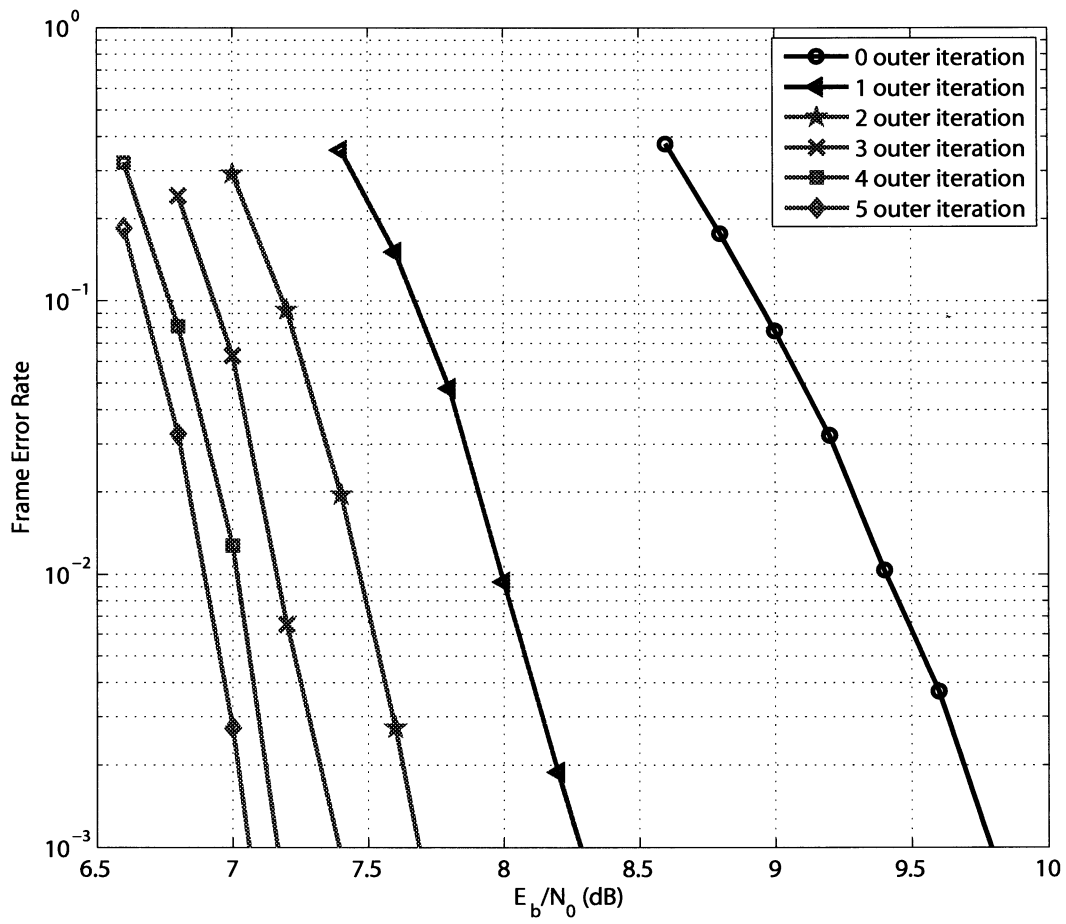


Figure 3.13 : Error performance of an iterative detection and decoding system, where a $M = 2$ trellis-search max-log-MAP detector is used.

3.4 VLSI Architecture for The Trellis-Search Detector

In this section, we describe VLSI architectures for the proposed PPTS detector. We introduce a fully-parallel “systolic” architecture to achieve the maximum throughput performance, and a “folded” architecture to reduce area for lower throughput application. For the sake of clarity, we describe a PPTS detector architecture with $M = 2$ for the 4×4 16-QAM system. It should be noted that the architecture described can be easily scaled for other values of M and other MIMO configurations.

3.4.1 Fully-Parallel Systolic Architecture

Fig. 3.14 shows the fully-parallel “systolic” architecture for a $N_t = 4$ antenna system. This architecture is fully pipelined so that it can process one MIMO symbol in every clock cycle. In this architecture, the main processing elements include 3 path reduction units (PRUs), 3 path extension units (PEUs), 4 path selection units (PSUs), and 4 LLR calculation (LLRC) units. The detailed structures of these processing elements will be described in the following subsections.

In Fig. 3.14, three PRUs (PRU₃₋₁) and one PSU (PSU₀) are employed to implement the path reduction algorithm. The main diagonal of the systolic array is related to the path reduction data flow shown in Fig. 3.2. The PRU implements one main iteration loop of Algorithm 1 by employing Q path reduction processors to simultaneously process Q nodes in a certain stage (cf. Fig. 3.2). PSU₀ implements the *final selection* step of Algorithm 1 by using Q search units. The data flow for

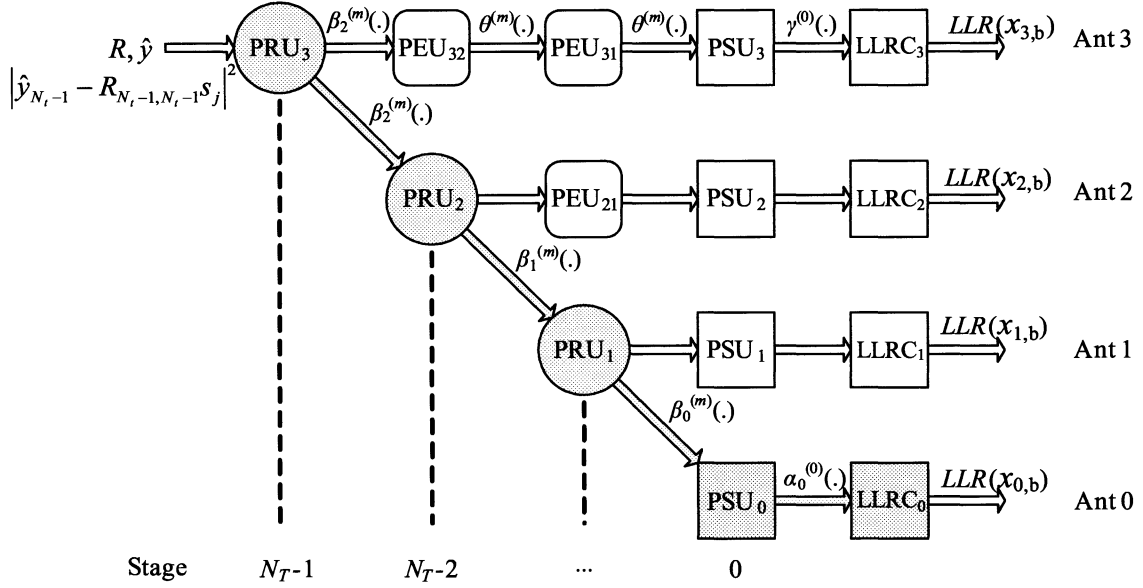


Figure 3.14 : A pipelined fully-parallel “systolic” architecture for the PPTS detector, where each PRU/PEU/PSU is a cluster of Q path reduction/path extension/path selection processors.

the path reduction is as follows. Firstly, PRU_3 receives \mathbf{R} , $\hat{\mathbf{y}}$, and the pre-computed $|\hat{y}_3 - R_{3,3}s_j|^2$, and it computes all the path candidates $\beta_2^{(m)}(i, j)$ in parallel, which are fed to the next PRU, i.e. PRU_2 . Then, PRU_2 computes $\beta_1^{(m)}(i, j)$, which are fed to PRU_1 , and so forth. Finally, PSU_0 receives $\beta_0^{(m)}(i, j)$ from PRU_1 and computes $\alpha_0^{(0)}(i)$, which are fed to LLRC_0 to compute $LLR(x_{0,b})$ based on (3.3).

In Fig. 3.14, three PEUs and three PSUs (PSU_{3-1}) are employed to implement the path extension algorithm. Each row (but the last) of the systolic array is related to the path extension data flow shown in Fig. 3.4. The PEU implements one main iteration loop of Algorithm 2 by employing Q path extension processors to simultaneously extend Q nodes in a certain stage (cf. Fig. 3.4). The PSU is used to implement the

final selection step of Algorithm 2. The data flow for the path extension is as follows.

To detect antenna $k \geq 1$, $k - 1$ number of the PEUs and 1 PSU are used. Let $t = k - 1$. Firstly, $\text{PEU}_{k,t}$ receives $\beta_{k-1}^{(m)}(i, j)$ from PRU_k and it computes $\theta^{(m)}(k, i, t - 1, j)$, which are fed to $\text{PEU}_{k,t-1}$. Next, $\text{PEU}_{k,t-1}$ computes $\theta^{(m)}(k, i, t - 2, j)$, which are fed to $\text{PEU}_{k,t-2}$, and so forth. Finally, PSU_k receives $\theta^{(m)}(k, i, 0, j)$ from $\text{PEU}_{k,1}$ and computes $\gamma^{(0)}(k, i, 0)$, which are fed to LLRC_k to compute $\text{LLR}(x_{k,b})$ based on (3.4). Note that to detect antenna 1, only one PSU (PSU_1) is required.

3.4.2 Path Reduction Unit (PRU)

The structure of the PRU is shown in Fig. 3.15. The PRU is used to implement the path reduction algorithm (cf. Algorithm 1:main loop). The PRU employs $Q = 16$ path reduction processors to process all the Q nodes in a certain stage in parallel. Each path reduction processor contains one minimum (min) finder unit (MFU) and one path calculation unit (PCU), where the MFU is used to select the best M paths $\alpha_k^{(m)}(i)$ from the QM incoming path candidates $\beta_k^{(m)}(j, i)$ (cf. Algorithm 1-1.a), and the PCU is used to compute the QM new extended path candidates $\beta_{k-1}^{(m)}(i, j)$ (cf. Algorithm 1-1.b).

Min Finder Unit (MFU)

The MFU is used to select the best $M = 2$ paths from $QM = 32$ path candidates. Fig. 3.16 shows the block diagram for the MFU unit which finds the minimum value Z_0 and the second minimum value Z_1 from its 32 data inputs (I_0 to I_{31}). The MFU

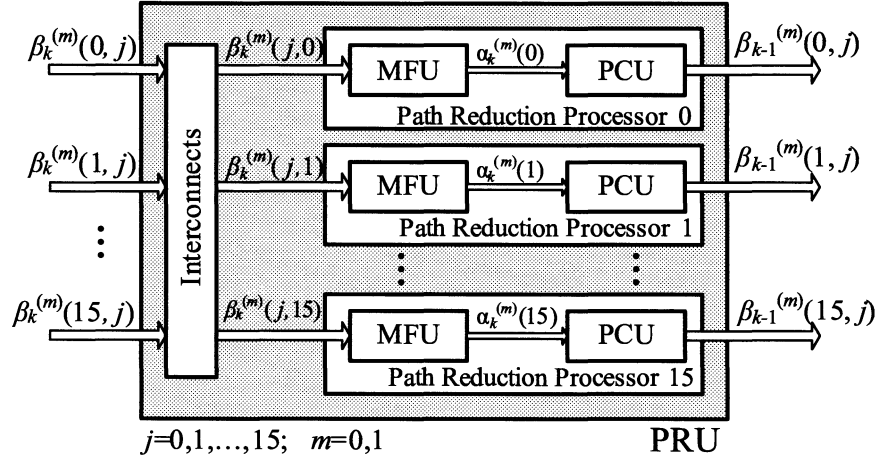


Figure 3.15 : Block diagram for the PRU, which contains $Q = 16$ path reduction processors.

comprises of 16 CMP (comparison) units, 15 variable size ($p : (p/2 + 1)$) C-S (compare and select) units, and one MIN unit. The structures of the CMP unit is shown in Fig. 3.17(a). The CMP unit compares two data inputs A and B , and outputs the smaller one (or the “winner”): $W = \min(A, B)$, and the larger one (or the “loser”): $L = \max(A, B)$, and the sign: $S = \text{sign}(A - B)$. The variable size $p : (p/2 + 1)$ C-S unit has p inputs ($A, U_1, U_2, \dots, U_{p/2-1}, B, V_1, V_2, \dots, V_{p/2-1}$) and $p/2 + 1$ outputs ($W, L_1, L_2, \dots, L_{p/2}$). The different values of p for the variable size C-S unit are 4, 6, 8, ..., $2 \log(QM)$. Output W of the C-S unit is the smallest data among all the p inputs. Outputs $L_1, L_2, \dots, L_{p/2}$ of the C-S unit are $p/2$ candidates for the second smallest data among all the p inputs. Fig. 3.17(b)(c) show the structures for the 4:3 C-S unit and the 6:4 C-S unit. The structures for the larger size C-S units, e.g. 8:5 C-S unit and 10:6 C-S unit, are omitted in this thesis because they have very similar structures as the 6:4 C-S unit.

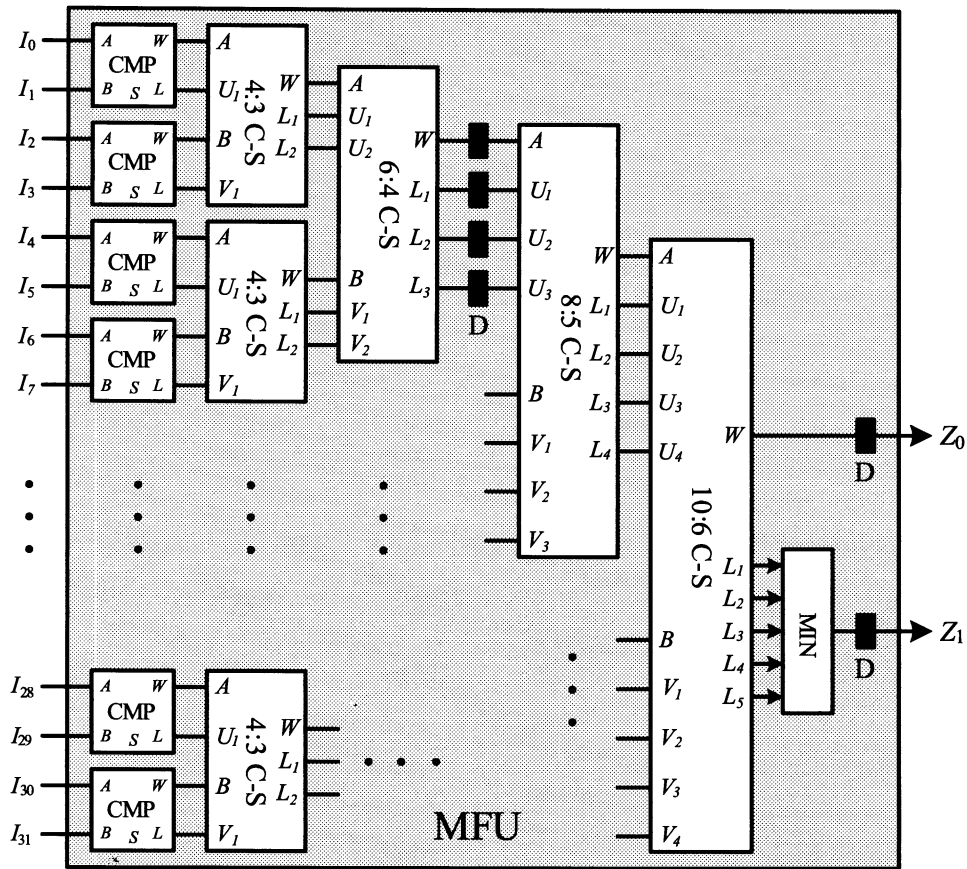


Figure 3.16 : Block diagram for the MFU, which uses 16 CMP units, 15 variable size C-S (compare and select) units, and 1 MIN unit to implement the (2,32) sorting.

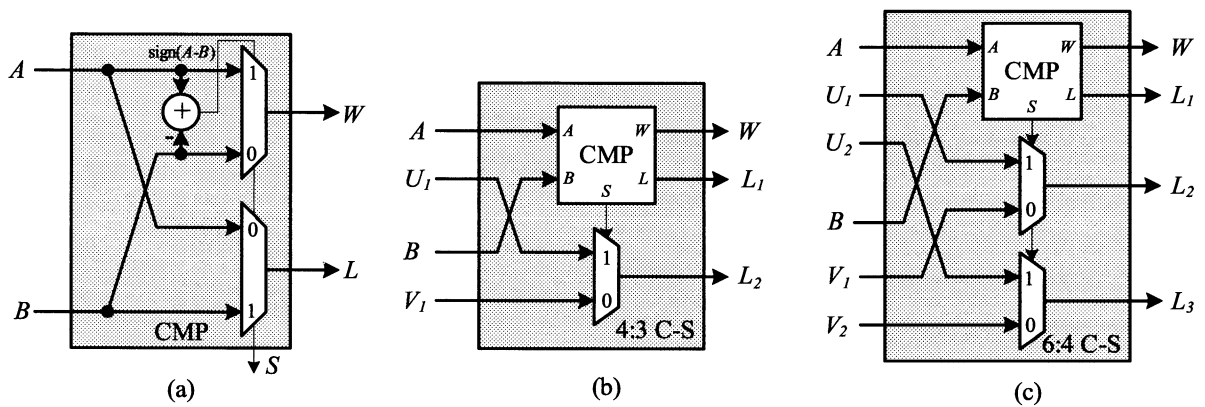


Figure 3.17 : Block diagram for the CMP unit, the 4:3 C-S unit, and the 6:4 C-S unit.

The MFU functions as follows. As shown in Fig. 3.16, the MFU takes $QM = 32$ data inputs and feeds them to 16 CMP units, where each CMP unit generates the winner and the loser of its two data inputs. The connection of the computational blocks in the MFU resembles a tree-like structure. Every two CMP units are connected to one 4:3 C-S unit, where the outputs of the 4:3 C-S unit are the winner (W) of its four data inputs, and two candidates (L_1, L_2) for the second winner. Every two 4:3 C-S units are connected to one 6:4 C-S unit, where the outputs of the 6:4 C-S unit are the smallest data (W) among its 6 data inputs, and three candidates (L_1, L_2, L_3) for the second smallest data. Similarly, every two 6:4 C-S units are connected to one 8:5 C-S unit, and two 8:5 C-S units are connected to a final 10:6 C-S unit. Finally, output W of the 10:6 C-S unit is the smallest data (Z_0) among the 32 data (I_0, I_1, \dots, I_{31}). Outputs L_1, L_2, \dots, L_5 of the 10:6 C-S unit are the five candidates for the second smallest data among the 32 data inputs. A MIN unit is used to generate the second smallest data Z_1 ($Z_1 = \min(L_1, L_2, \dots, L_5)$).

Path Calculation Unit (PCU)

Fig. 3.18 shows the PCU architecture which employs $M = 2$ partial Euclidean distance calculation (PEDC) units to compute $QM = 32$ path metrics in parallel. The partial Euclidean distance (PED) d_{k-1} is computed recursively as

$$d_{k-1} = d_k + e_{k-1}. \quad (3.15)$$

The metric increment e_{k-1} (cf. (3.1)) is computed as follows:

$$e_{k-1} = |T + R_{k-1,k-1} \cdot s_{k-1}|^2, \quad (3.16)$$

where

$$T = \sum_{j=k}^{N_T-1} R_{k-1,j} \cdot s_j - \hat{y}_{k-1}. \quad (3.17)$$

For a given PED d_k , we need to compute $Q = 16$ new PEDs d_{k-1} . Instead of computing each new PED separately, we can compute Q new PEDs in a group by knowing that symbol s_{k-1} is drawn from a known alphabet: $s_{k-1} \in \{\pm 1 \pm j, \pm 1 \pm 3j, \pm 3 \pm j, \pm 3 \pm 3j\}$, and $R_{k-1,k-1}$ is a real value if using a certain QR decomposition method, e.g. Gram-Schmidt QR decomposition [88]. Let $s_{k-1}(q)$, $q = 0, 1, \dots, Q-1$, denote the complex symbol for the q -th constellation point in the alphabet. Then (3.16) is re-expressed as:

$$e_{k-1}(q) = |T|^2 + R_{k-1,k-1}^2 |s_{k-1}(q)|^2 + 2 \operatorname{Re} \left((R_{k-1,k-1} \cdot T^*) s_{k-1}(q) \right). \quad (3.18)$$

We pre-compute $R_{k-1,k-1}^2 |s_{k-1}(q)|^2$ for different q and save them in registers. Fig. 3.19 shows the architecture for the PEDC unit, which computes $Q = 16$ PEDs in parallel. In this architecture, a shift and add (SHAD) unit is used to implement the constant multiplication $A \cdot s_{k-1}$, a multiplier (MULT) is used to implement $R_{k-1,k-1} \cdot T^*$, and a CPX NORM unit is used to compute the L2 norm ($|T|^2$) of the complex signal T .

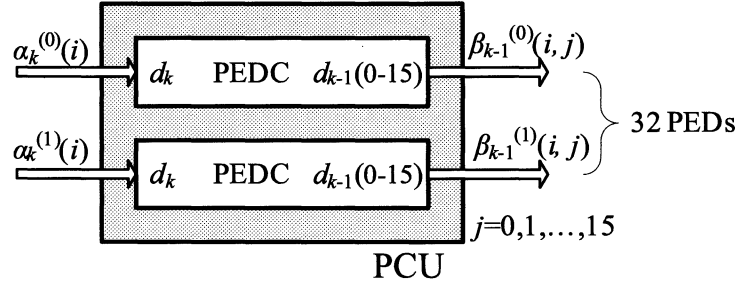


Figure 3.18 : Block diagram for the PCU, which employs $Q = 2$ PEDC units.

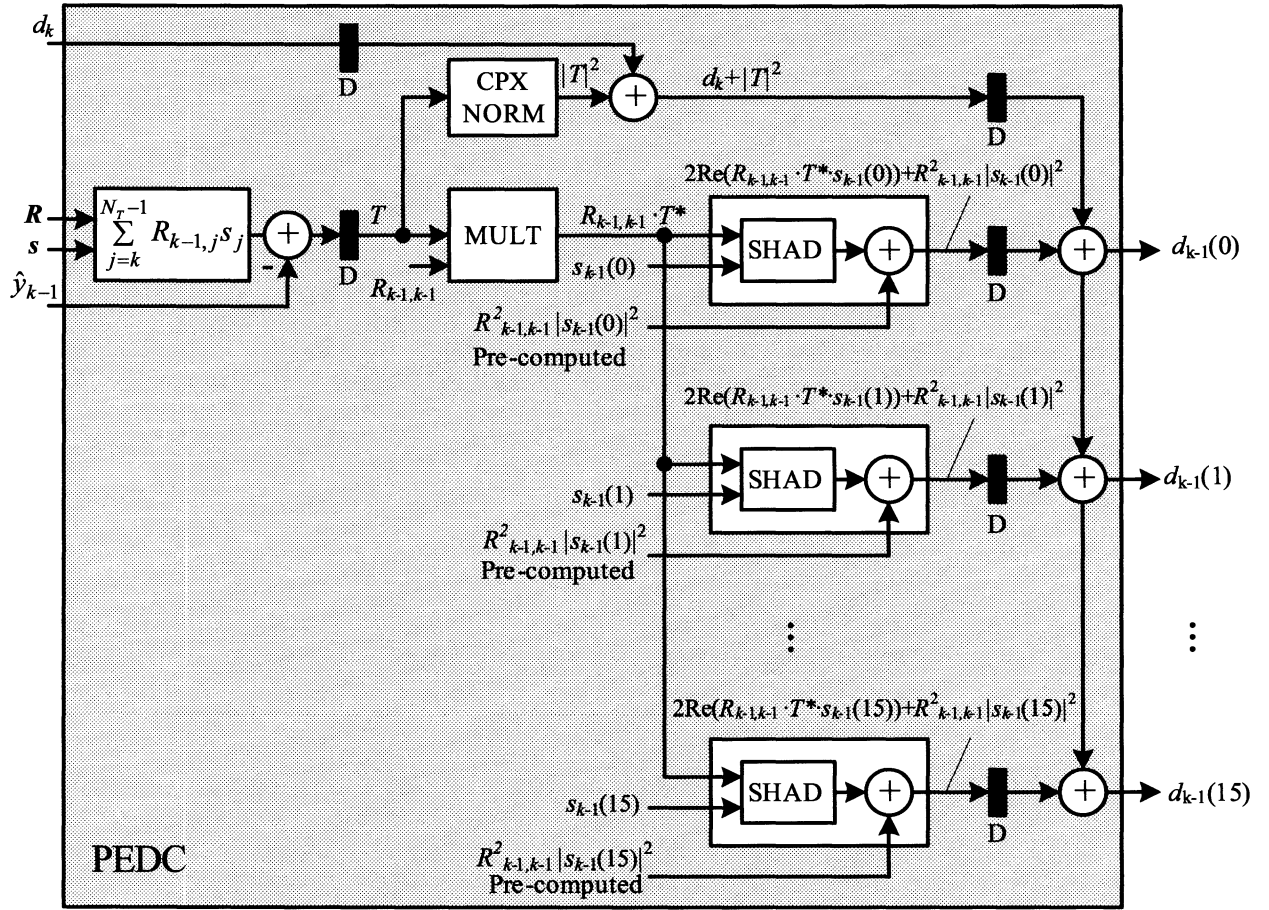


Figure 3.19 : Block diagram for the PEDC unit, which computes 16 PEDs in parallel.

3.4.3 Path Extension Unit (PEU)

The PEU implements the path extension algorithm (cf. Algorithm 2:main loop). The PEU has a very similar architecture to the PRU. Fig. 3.20 shows the block diagram for the PEU, which employs $Q = 16$ path extension processors to extend Q nodes in a certain stage in parallel. Each path extension processor contains one MFU and one PCU, where the MFU is used to select the best M paths $\gamma^{(m)}(k, i, t)$ from QM path candidates $\theta^{(m)}(k, i, t, j)$ (cf. Algorithm 2-1.a), and the PCU is used to calculate the QM new extended path candidates $\theta^{(m)}(k, i, t-1, j)$ (cf. Algorithm 2-1.b)

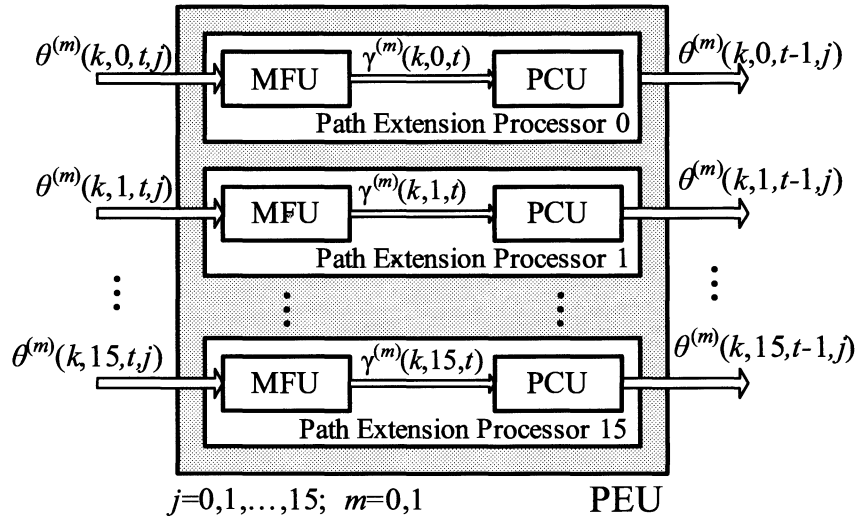


Figure 3.20 : Block diagram for the PEU, which contains $Q = 16$ path extension processors.

3.4.4 Path Selection Unit (PSU)

The PSU implements the *final selection* step in Algorithm 1 or Algorithm 2. As shown in Fig. 3.21, the PSU contains only Q MFUs to realize Q concurrent sorting

(M, QM) .

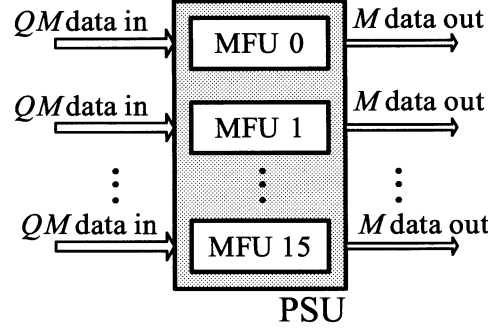


Figure 3.21 : Block diagram for the PSU, which contains $Q = 16$ MFUs.

3.4.5 LLR Computation Unit (LLRC)

The LLRC is used to compute LLRs based on (3.3) or (3.4). Fig. 3.22 shows the block diagram of the LLRC unit. To compute $\log_2(Q) = 4$ LLRs for antenna k in parallel, we need 4 sets of hardware blocks shown in Fig. 3.22 to compute $LLR(x_{k,b})$, $b = 0, 1, \dots, \log Q - 1$, for our example 16-QAM system. It should be noted that the multiplier in Fig. 3.22 may not be required if the outer channel decoder uses a linear decoding algorithm such as the Min-Sum algorithm [63] in LDPC decoding or the Max-Log-MAP algorithm [89] in Turbo decoding. In that case, the multiplier can be replaced by a simpler normalizer. To support the n-Term-Log-MAP algorithm, the LLRC block needs to be modified by replacing the MIN unit with a n-input Log-sum unit. Fig. 3.23 shows an example for the eight-term log-sum unit.

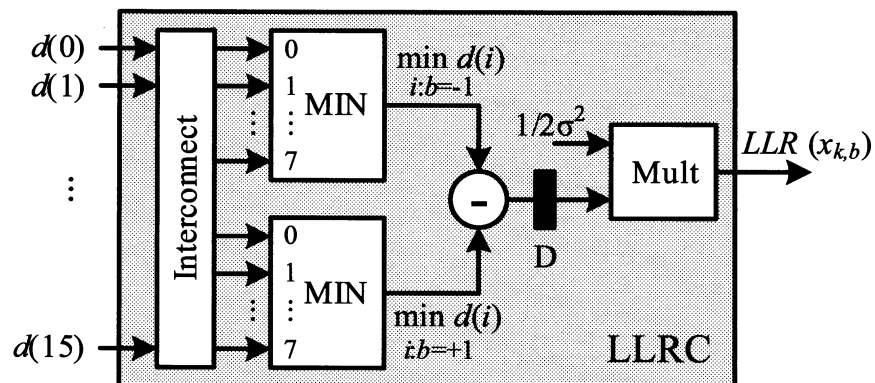


Figure 3.22 : Block diagram of the LLRC unit.

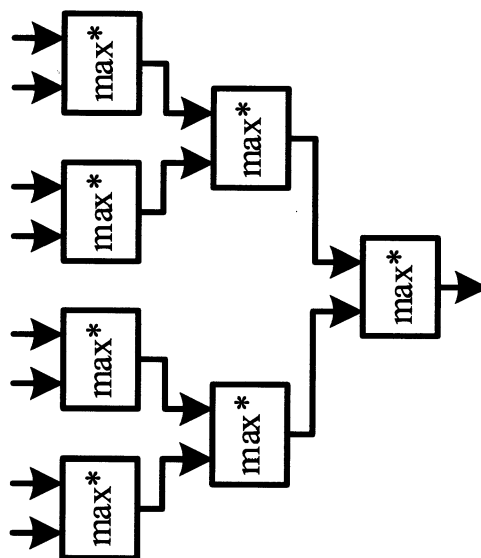


Figure 3.23 : Eight-term log-sum unit.

3.4.6 Throughput Performance of The Systolic Architecture

The proposed systolic MIMO detector architecture (cf. Fig. 3.14) can provide very high throughput performance. This architecture is fully pipelined so that it can process one MIMO symbol in every clock cycle. Generally, if we let the clock frequency be clk MHz, then the throughput (Mbps) for a $N_t \times N_r$ Q -QAM system can be expressed as:

$$\text{Throughput}_{\text{Systolic}} = N_t \cdot \log_2 Q \cdot clk. \quad (3.19)$$

As an example, assuming a system clock of 400 MHz, the systolic architecture can provide a throughput of 6.4 Gbps for a 4×4 16-QAM system.

3.4.7 Folded Architecture

For system applications that may require less throughput, we can fold the fully-parallel systolic architecture to reduce the parallelism and hence the hardware complexity. Fig. 3.24 shows the folded architecture where only one PRU and one PEU are instantiated to save area. Note that the PRU/PEU is the most area-consuming block in the PPTS detector.

Because we only have one PRU and one PEU, we need to schedule them sequentially. Fig. 3.25 illustrates the detection timing diagram using the folded architecture for a 4 antenna system. In this diagram, the PRU is scheduled to run the path reduction (PR) operations from $t=0$ to $t=11$, and the PEU is scheduled to run the path extension (PE) operations from $t=4$ to $t=15$. Note that the subscripts of the PRs

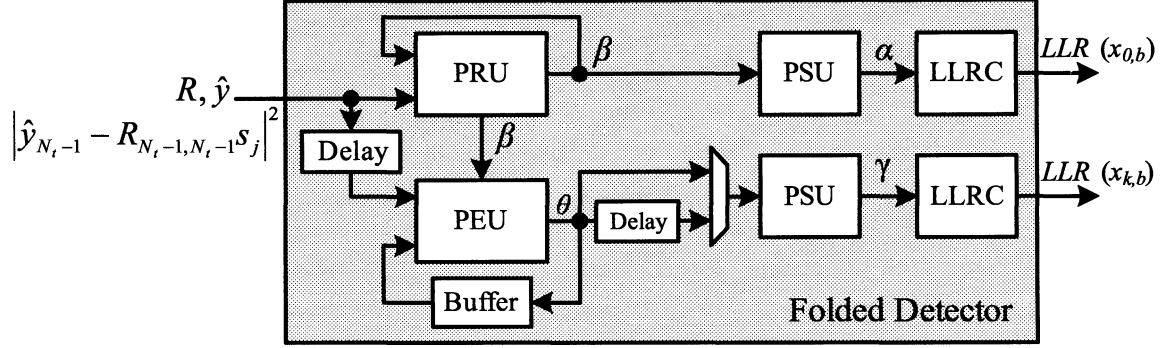


Figure 3.24 : Folded architecture for the PPTS detector.

and PEs in this diagram have the same meaning as that in Fig. 3.14. For simplicity, the final path selection operations (executed in PSU) and the LLR calculation operations are omitted in this diagram. Furthermore, as the pipeline stages for the PRU and PEU are 4 clock cycles, we can feed four back-to-back MIMO symbols in 4 consecutive cycles, e.g at $t, t+1, t+2, t+3$ to fully utilize the hardware. And we can feed the next four back-to-back MIMO symbols at $t+12, t+13, t+14, t+15$ into the pipeline, and so forth. The throughput of the folded architecture for a 4 antenna system is given as:

$$\text{Throughput_folded_4ant} = \frac{4}{3} \log_2 Q \cdot fclk. \quad (3.20)$$

For a larger MIMO system with $N_t \geq 4$ transmit antennas, if we still use one PRU and one PEU, the throughput for a $N_t \geq 4$ antenna system is estimated as:

$$\text{Throughput_folded_N} = \frac{2N_t}{(N_t - 1)(N_t - 2)} \log_2 Q \cdot fclk. \quad (3.21)$$

As an example, assuming a system clock of 400 MHz, the systolic architecture can

provide a throughput of 2.13 Gbps for a 4×4 16-QAM system. As a balanced tradeoff, the folded architecture significantly reduce the area but still maintaining high throughput performance.

Note that for larger MIMO systems ($N_t > 4$), the throughput is limited by the number of the path extension operations. However, we can employ more than one PEU in the folded architecture to match with the processing speed of the PRU.

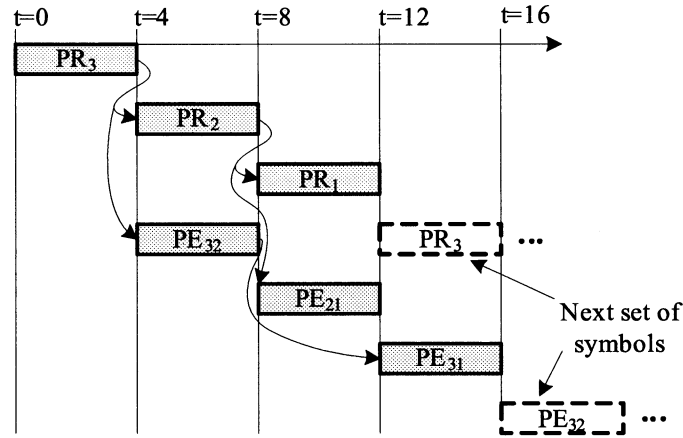


Figure 3.25 : Detection timing diagram for a 4 antenna system using the folded architecture.

3.5 Summary

In this chapter, we introduce a novel low-complexity trellis-search detection algorithm and VLSI architecture. In chapter 6, we will describe an ASIC implementation of a multi-Gbps MIMO detector based on this trellis-search architecture. In this chapter, we also introduce an iterative detection decoding scheme which can be used to improve the error performance of the MIMO system by around 3 dB through the use of the

proposed PPTS detection approach. In chapter 4 and 5, we will describe two kinds of channel decoders (Turbo decoders and LDPC decoders) that can be used to integrate with the MIMO detector to form an iterative receiver.

Chapter 4

High-Throughput Turbo Detector for LTE/LTE-Advanced System

Turbo codes invented in 1993 [47] have attracted much attention recently because the new wireless systems are demanding higher and higher data rate. For example, in the LTE-Advanced standard, the target data rate is 1 Gbps, which poses a significant challenge for the Turbo decoder design. Our goal is to develop a highly-parallel Turbo decoder architecture to achieve 1+ Gbps high data rate. We utilize the contention-free interleaver defined in the LTE standard to enable parallel Turbo decoding without additional data buffers.

Turbo decoders suffer from high decoding latency due to the iterative decoding process, the forward-backward recursion in the maximum *a posteriori* (MAP) decoding algorithm and the interleaving/de-interleaving between iterations [47, 90, 91]. Sliding window architectures are often used to reduce the latency of the MAP decoding. The choice of the sliding window algorithm may have a significant impact on the decoding BER performance and parallelism. In this chapter, we will present a new parallel sliding window algorithm and a new parallel non-sliding window algorithm for the LTE Turbo decoding.

A high throughput Turbo decoder can be realized by parallelizing several MAP

decoders, where each MAP decoder operates on a segment of the received codeword [92]. Due to the randomness of the Turbo interleaver, two or more MAP decoders may access the same memory at the same clock cycle which will lead to a memory collision. As a result, the decoder has to be stalled which consequently delays the decoding process. The Interleaver structures in the 3G standards, such as CDMA/W-CDMA/UMTS, do not have a parallel structure. Although the memory stalls caused by the interleaver can be partially reduced by using write buffers [93], the memory stalls will occur more and more frequently as the parallelism degree increases. To solve this problem, the high data rate 3GPP LTE standard has adopted a contention-free, parallel interleaver which is called quadratic permutation polynomial (QPP) Turbo interleaver [94]. From an algebraic-geometric perspective, the QPP interleaver allows analytical designs and simplifies hardware implementation of a parallel Turbo decoder [95]. Based on the permutation polynomials over integer rings, every factor of the interleaver length can be a parallelism degree for the decoder [95] which is contention-free.

Turbo decoder architectures in the literature are mostly based on the older matrix-permutation interleavers, thus the parallelism level is significantly limited. In this chapter, we will utilize the conflict-free QPP interleaving property to design a highly-parallel Turbo decoder for high speed wireless applications. The proposed decoder can achieve over 1Gbps data rate, which is significantly higher than the existing Turbo decoders.

4.1 LTE/LTE-Advanced Turbo Codes

As shown in Figure 4.1, the Turbo encoding scheme in the LTE/LTE-Advanced standard is a parallel concatenated convolutional code with two 8-state constituent encoders and one quadratic permutation polynomial (QPP) interleaver [94]. The function of the QPP interleaver is to take a block of N -bit data and produce a permutation of the input data block. From the coding theory perspective, the performance of a Turbo code depends critically on the interleaver structure [49]. The basic LTE Turbo coding rate is $1/3$. It encodes an N -bit information data block into a codeword with $3N + 12$ data bits, where 12 tail bits are used for trellis termination. The initial value of the shift registers of the 8-state constituent encoders shall be all zeros when starting to encode the input information bits. LTE has defined 188 different block sizes, $40 \leq N \leq 6144$.

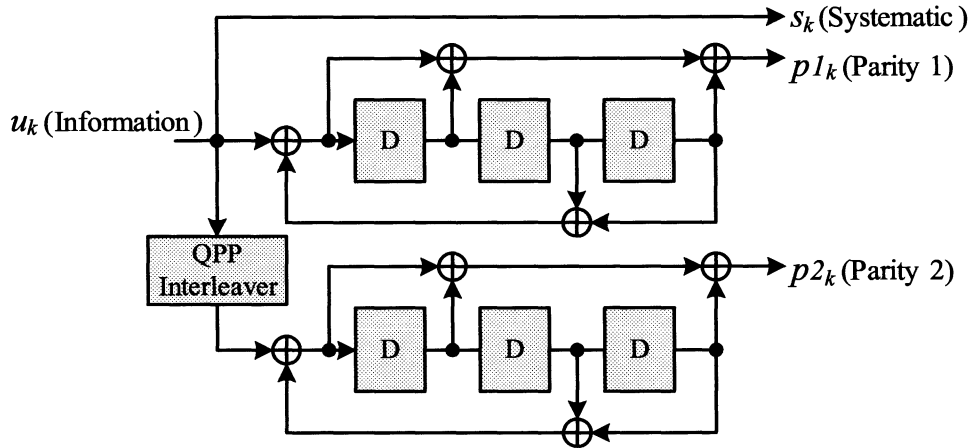


Figure 4.1 : Structure of rate 1/3 Turbo encoder in the LTE/LTE-advanced system.

4.2 QPP Interleaver

The task of an interleaver is to permute the soft values generated by the MAP decoder and write them into random or pseudo-random positions. Interleaving/deinterleaving of extrinsic information is a key issue that needs to be addressed to enable parallel decoding because memory access contention may occur when MAP decoders fetch/write extrinsic information from/to memory. The QPP interleaver defined in the new LTE/LTE-advanced standard differs from previous 3G interleavers in that it is based on algebraic constructions via permutation polynomials over integer rings. It is known that permutation polynomials generate contention-free interleavers [96, 95], i.e. every factor of the interleaver length becomes a possible parallelism degree.

4.2.1 Algebraic Description of QPP Interleaver

The QPP interleaver can be expressed via a simple mathematical formula. Given an information block length N , the x -th interleaving output position is specified by the quadratic expression: [94]

$$f(x) = (f_2x^2 + f_1x) \bmod N, \quad (4.1)$$

where parameters f_1 and f_2 are integers and depend on the block size N ($0 \leq x, f_1, f_2 < N$). For each block size, a different set of parameters f_1 and f_2 are defined. In LTE, all the block sizes are even numbers and are divisible by 4 and 8. Moreover, the block size N is always divisible by 16, 32, and 64 when $N \geq 512$, $N \geq 1024$, and $N \geq 2048$, respectively. By definition, parameter f_1 is always an odd number

whereas f_2 is always an even number. Through further inspection, we can list the following algebraic properties for the QPP interleaver.

QPP interleaver algebraic property 1:

$f(x)$ has the same even/odd parity as x :

$$f(2k) \bmod 2 = 0$$

$$f(2k + 1) \bmod 2 = 1.$$

QPP interleaver algebraic property 2:

The remainders of $f(x)/4$, $f(x + 1)/4$, $f(x + 2)/4$, and $f(x + 3)/4$ are unique:

$$f(4k) \bmod 4 = 0$$

$$f(4k + 1) \bmod 4 = \begin{cases} 1 & \text{when } (f_1 + f_2) \bmod 4 = 1 \\ 3 & \text{when } (f_1 + f_2) \bmod 4 = 3 \end{cases}$$

$$f(4k + 2) \bmod 4 = 2$$

$$f(4k + 3) \bmod 4 = \begin{cases} 3 & \text{when } (f_1 + f_2) \bmod 4 = 1 \\ 1 & \text{when } (f_1 + f_2) \bmod 4 = 3. \end{cases}$$

QPP interleaver algebraic property 3:

$$f(x) \bmod n = f(x + m) \bmod n, \forall m : m \bmod n = 0.$$

Property 1 can be easily verified since parameter f_2 is always even and parameter f_1

is always odd by definition. Property 2 can be shown through the following equations:

$$\begin{aligned}
 f(4k) &= 4(4f_2k^2 + f_1k) \\
 f(4k+1) &= 4(4f_2k^2 + 2f_2k + f_1k) + f_2 + f_1 \\
 f(4k+2) &= 4(4f_2k^2 + 4f_2k + f_1k + f_2) + 2f_1 \\
 f(4k+3) &= 4(4f_2k^2 + 6f_2k + f_1k + 2f_2) + f_2 + 3f_1.
 \end{aligned}$$

Property 3 can be verified by:

$$f(x+m) = f(x) + m(2f_2x + f_2m + f_1).$$

We will explain later that these algebraic properties are very useful in designing memory systems for parallel Turbo decoder.

4.2.2 QPP Contention-Free Property

In general, a Turbo interleaver/de-interleaver $f(x)$, is said to be contention-free for a window size of L if and only if it satisfies the following constraint [95, 97, 98]

$$\left\lfloor \frac{f(x+iL)}{L} \right\rfloor \neq \left\lfloor \frac{f(x+jL)}{L} \right\rfloor, \quad (4.2)$$

where $0 \leq x < L$, $0 \leq i, j < P (= N/L)$, and $i \neq j$. The terms in (4.2) are essentially the memory indices that are concurrently accessed by the P MAP decoder cores. If these memory indices are unique during each read and write operation, then there are no contentions in memory accesses. Figure 4.2 shows an example of the contention-free memory access scheme.

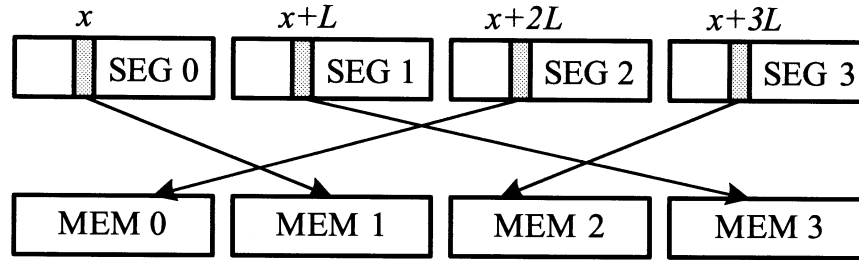


Figure 4.2 : An example of the contention-free interleaving, where a data block is divided into $P = 4$ segments (SEG 0 to SEG 3) with equal length of $L = N/P$. The contention-free property requires that for a fixed offset x at each segment, the segment indices for the interleaving addresses $\left\lfloor \frac{f(x+iL)}{L} \right\rfloor$ ($0 \leq i \leq P - 1$) are unique so that they can be physically mapped to different memory modules.

It has been shown in [96, 95] that every factor of the interleaver length N becomes a possible interleaver parallelism that satisfies the contention-free requirement in (4.2).

Table 4.1 summaries the parallelism degrees (up to 64) for some of the LTE QPP interleavers.

Table 4.1 : QPP interleaver parallelism.

N	$f(x)$	Parallelism (factors of N)
40	$10x^2 + 3x$	1,2,4,5,8,10,20
48	$12x^2 + 7x$	1,2,3,4,6,8,12,16,24
64	$42x^2 + 19x$	1,2,4,8,16,32
...
6016	$94x^2 + 23x$	1,2,4,8,16,32,47,64
6080	$190x^2 + 47x$	1,2,4,5,8,10,16,19,20,32,38,40,64
6144	$480x^2 + 263x$	1,2,3,4,6,8,12,16,24,32,48,64

4.2.3 Hardware Implementation of QPP Interleaver

Based on the algebra analysis in [96], the QPP interleaver is guaranteed to always generate a unique address which greatly simplifies the hardware implementation. In MAP trellis decoding, the QPP interleaving addresses are usually generated in a consecutive order (with step size of d). By taking advantage of this fact, the QPP interleaving address can be computed in a recursive manner. Suppose the interleaver starts at x_0 , we first pre-compute $f(x_0)$ as:

$$f(x_0) = (f_2x_0^2 + f_1x_0) \bmod N. \quad (4.3)$$

In the following cycles, as x is incremented by d , $f(x + d)$ is computed recursively as follows:

$$f(x + d) = (f_2(x + d)^2 + f_1(x + d)) \bmod N \quad (4.4)$$

$$= (f(x) + g(x)) \bmod N, \quad (4.5)$$

where $g(x)$ is defined as:

$$g(x) = (2df_2x + d^2f_2 + df_1) \bmod N. \quad (4.6)$$

Note that $g(x)$ can also be computed in a recursive manner:

$$g(x + d) = (g(x) + 2d^2f_2) \bmod N \quad (4.7)$$

$$= (g(x) + (2d^2f_2 \bmod N)) \bmod N. \quad (4.8)$$

The initial value $g(x_0)$ needs to be pre-computed as:

$$g(x_0) = (2df_2x_0 + d^2f_2 + df_1) \bmod N. \quad (4.9)$$

The modulo operation in (4.5) and (4.8) can be difficult to implement in hardware if the operands are not known in advance. However, by definition we know that both $f(x)$ and $g(x)$ are less than N , and parameters f_1 and f_2 are both less than N too. Thus, the modulo operations in (4.5) and (4.8) can be simply realized by additions and subtractions. In the LTE standard, the value N is between 40 and 6144.

In the proposed method, three numbers need to be pre-computed: $(2d^2f_2) \bmod N$, $f(x_0)$, and $g(x_0)$. Figure 4.3 shows a hardware architecture to compute the interleaving address $f(x)$, where x starts from x_0 and is incremented by d on every clock cycle. For example, by setting d to 1, this circuit can generate interleaving addresses at each step of 1. If n consecutive interleaving addresses are required at each clock cycle, this circuit can be replicated n times with n different initial values: x_0 , $x_0 + 1$, ..., and $x_0 + n - 1$.

The circuit in Figure 4.3 can generate interleaving address in a descending order as well by setting d to be a negative number, eg. $d = -1$. But $g(x_0)$ needs to be recomputed for negative d . To be able to generate both forward and backward addresses using the same $f(x)$ and $g(x)$ functions, we now describe a method to generate the QPP interleaving address in the descending order. By substituting x with $x - d$ in (4.5) and reorganize (4.5), we can get:

$$f(x - d) = (f(x) - g(x - d)) \bmod N. \quad (4.10)$$

Similarly, substitute x with $x - d$ in (4.8) and reorganize (4.8), we can get:

$$g(x - d) = (g(x) - (2d^2f_2 \bmod N)) \bmod N. \quad (4.11)$$

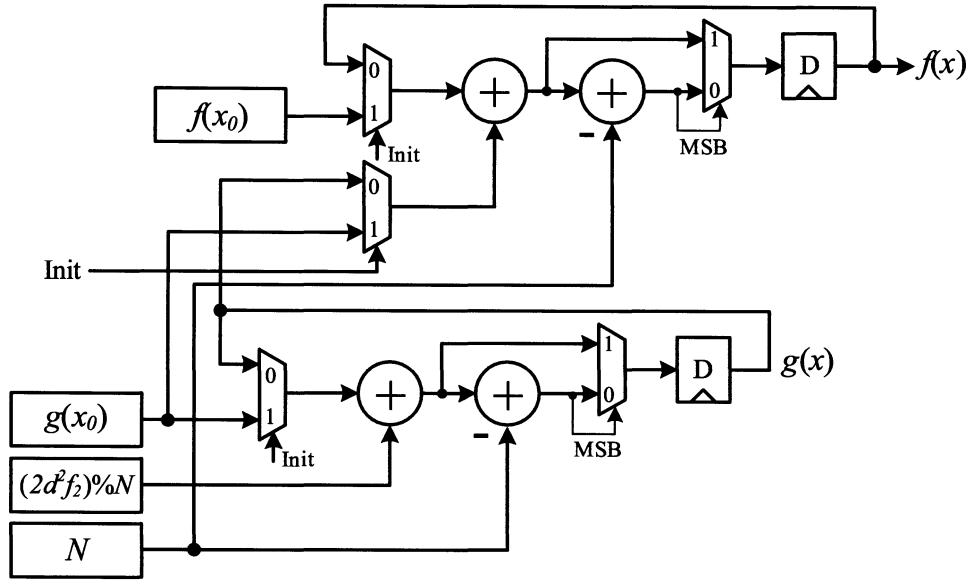


Figure 4.3 : Forward QPP address generator circuit diagram, step size = d .

Based on (4.10)(4.11), Figure 4.4 shows a hardware architecture to compute the QPP address $f(x)$ in the descending order (backward generating), where x starts from x_0 and is decremented by d on every clock cycle. The three pre-computed values are the same as those in the forward QPP address generator (cf. Figure 4.3).

As can be seen from Figure 4.3 and 4.4, the proposed QPP interleaver pattern generator consumes very few resources. The complexity of this circuit is an order of magnitude smaller than the previous 3G interleavers. For example, a circuit with about 30K gate count is reported in [99] to generate the interleaving addresses for Turbo codes in the previous 3G standard (3GPP Release-4), and a UMTS hardware interleaver with 10.5K gate count is presented in [100]. The low complexity of the proposed QPP interleaver is achieved due to the fact that the addresses are calculated

sequentially, not randomly.

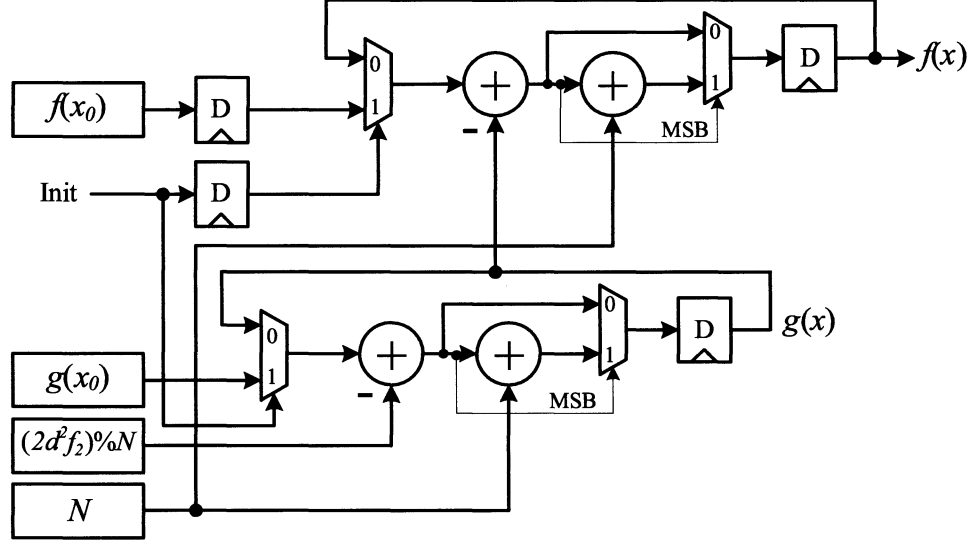


Figure 4.4 : Backward QPP address generator circuit diagram, step size = d .

4.3 Sliding Window and Non-Sliding Window MAP Decoder Architecture

MAP decoder architectures have been studied by many researchers [101, 102, 103, 104, 101, 105, 106]. Several factors, such as interleaver structure and sliding window scheme, must be considered when choosing an appropriate MAP decoder for LTE Turbo decoding. In this section we modify two low-latency MAP decoder architectures and propose a low-complexity QPP interleaving address generator to operate full-speed with the MAP decoder.

Due to the double recursion in the MAP decoding algorithm [91], the MAP decoder

suffers from high decoding latency. To reduce the decoding latency, the sliding window algorithm is often used [107]. However, the problem of the sliding window approach is the unknown backward (or forward) state metrics which are required in the beginning of the backward (or forward) recursion. We refer to the state metrics at sliding window length distance as *stakes*. These stakes can be estimated by using a training calculation [107], which will result in an additional decoding delay depending on the training length. For LTE Turbo codes, we do not recommend this traditional sliding window method when the Turbo coding rate is high. Because many parity bits will be removed after the base Turbo code is punctured to a higher code rate, the training length has to be increased to accurately estimate the state metrics at those stakes which consequently delays the decoding process.

For LTE Turbo decoding, we suggest to use a low-latency decoding method, referred to as state metric propagation (SMP) method, where the state metrics at stakes are initialized with stakes from the previous iteration [108]. In the very first iteration, uniform state metrics can be used for initialization. This method avoids the training calculation by propagating the state metrics to the next iteration. This method is especially useful when the Turbo coding rate is high. Based on our simulation results, the performance degradation caused by the window truncation in the SMP method is smaller than that in the traditional training based sliding window method in the case of high Turbo code rate. To compare the decoding performance using these two sliding window algorithms for high rate LTE Turbo codes, we perform floating point

simulations using BPSK modulation over AWGN channel. The LTE rate matching algorithm [94] is used for code puncturing. Figure 4.5 shows the floating-point simulation result for a rate of 0.95 Turbo code. Because of the high code rate, the maximum number of iterations is set to 10. In the figure, we show the block error rate (BLER) curves for the SMP based sliding window algorithm and the traditional training based sliding window algorithm. In the traditional training algorithm, we assume the training length is equal to the window length. As can be seen, the BLER performance of the SMP algorithm with window length $W = 64$ is better than that of the training algorithm with window length $W = 64$, and is close to that of the training algorithm with $W = 96$. The SMP algorithm with $W = 96$ and the training algorithm with $W = 128$ perform close to the optimal case when there is no window effect. Because of the good decoding performance and low decoding delay, we adopted the SMP algorithm in our Turbo decoder design.

The SMP based sliding window (SW) MAP algorithm (SW-MAP) has a window overhead of W (c.f. Figure 4.6(a)), which will lead to additional decoding delays. To eliminate this window overhead, we also consider a non-sliding window (NSW) based MAP algorithm (NSW-MAP) which is shown in Figure 4.6(b). To be more general, we consider the case of decoding a segment of the code block where the segment length is $L = N/P$. In the SW algorithm, a sliding window is applied to the backward recursion where the stakes are initialized from the previous Turbo iteration. If the window length is W , then $(L/W) \times 2$ stakes need to be saved (note that MAP

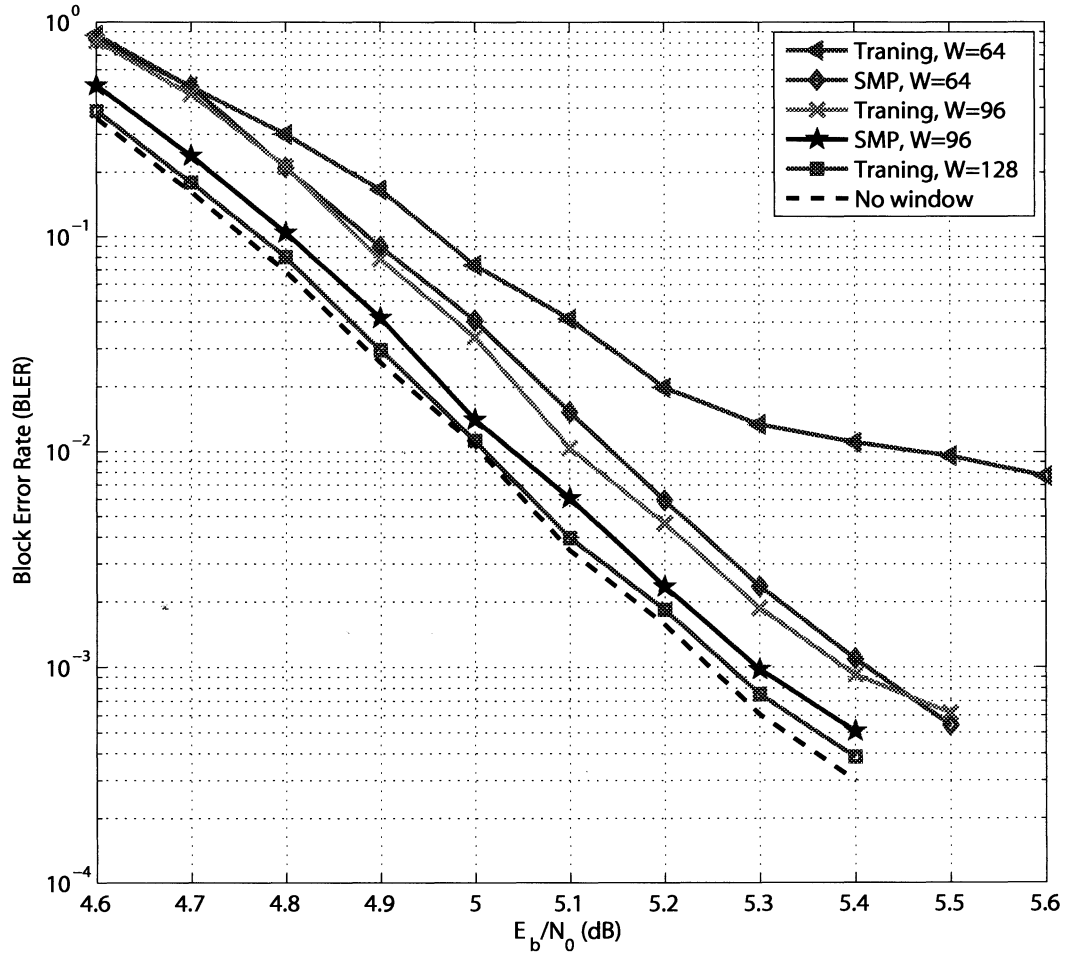


Figure 4.5 : Simulation result for a rate of 0.95 LTE Turbo code using two different sliding window algorithms.

1 can only be initialized with stakes from MAP 1, not from MAP 2, resulting in twice the amount of stake memory). In the NSW algorithm, no sliding window is applied to the backward recursions. So only the stakes at the end of the recursion needed to be saved. It should be noted that the memory bandwidth of the NSW-MAP algorithm is higher than the SW-MAP algorithm since two LLRs are read and two LLRs are written in one cycle. When the decoder parallelism is high, i.e. P is large, the NSW-MAP algorithm has throughput advantage over the SW-MAP algorithm. There are many other varieties of the MAP algorithms. See [109] for a thorough analysis of the MAP decoder architectures. In this thesis, we primarily focus on these two simple but effective MAP algorithms, and we will present QPP interleaving address generator architectures for these two MAP algorithms.

4.3.1 QPP Interleaving Address Generator for SW-MAP Decoder

Figure 4.7 shows the recommended SW-MAP decoder architecture. The SW-MAP decoder requires one set of α unit, β unit, branch unit, and LLRC unit because of the single flow structure. It employs fully parallel add-compare-select-add (ACSA) [110] units to calculate the state metrics in the α and β recursion processes. A SMP buffer was used to save the stakes for use in the next Turbo iteration. In the SW algorithm, the channel LLRs (systematic L_s and parity L_p) are loaded from the symbol memory in the sequential order. *A priori* information $LLR(in)$ are loaded from the LLR memory in the sequential order for the first half iteration, and in

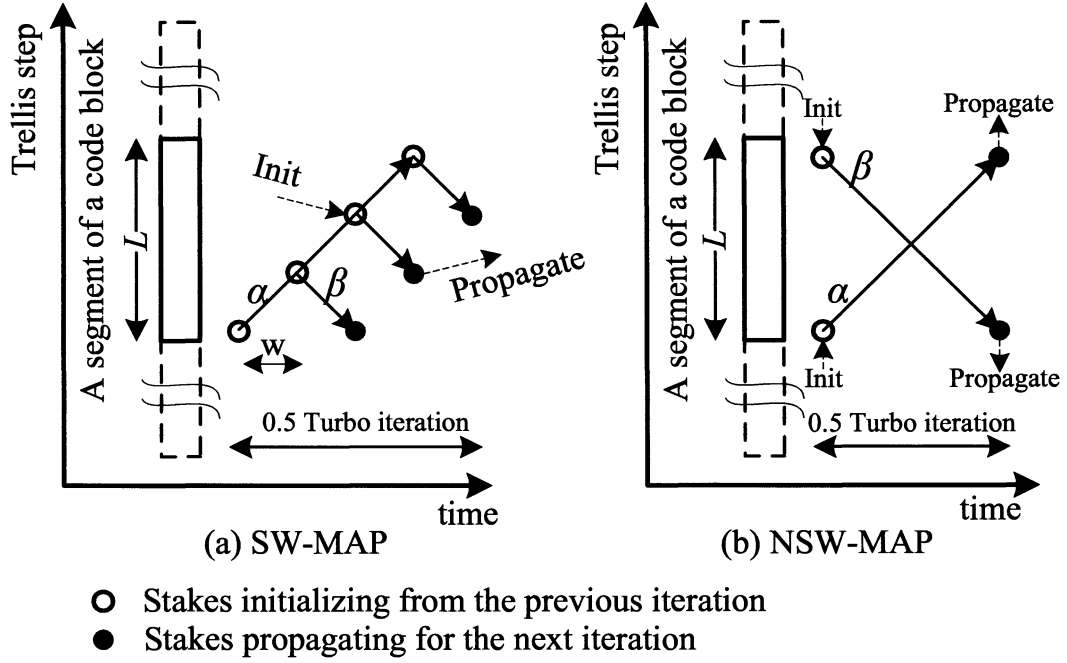


Figure 4.6 : Two recommended MAP decoding algorithms for LTE Turbo codes. (a) SW-MAP decoding algorithm. (b) NSW-MAP decoding algorithm.

the interleaving order for the second half iteration. The soft information $LLR(out)$ are written to the LLR memory in the backward sequential order during the first half iteration, and in the backward interleaving order for the second half iteration. To avoid loading interleaving systematic LLRs from the symbol memory during the second half iteration, we have modified the MAP algorithm to combine the systematic LLR with the extrinsic LLR in the first half iteration.

In this algorithm, the interleaving addresses must be generated during the second half iteration to provide read and write addresses to the LLR memory. In the SW algorithm, the read operation is in the forward direction, whereas the write operation is in the backward direction and is always behind the read operation. Figure 4.8(a)

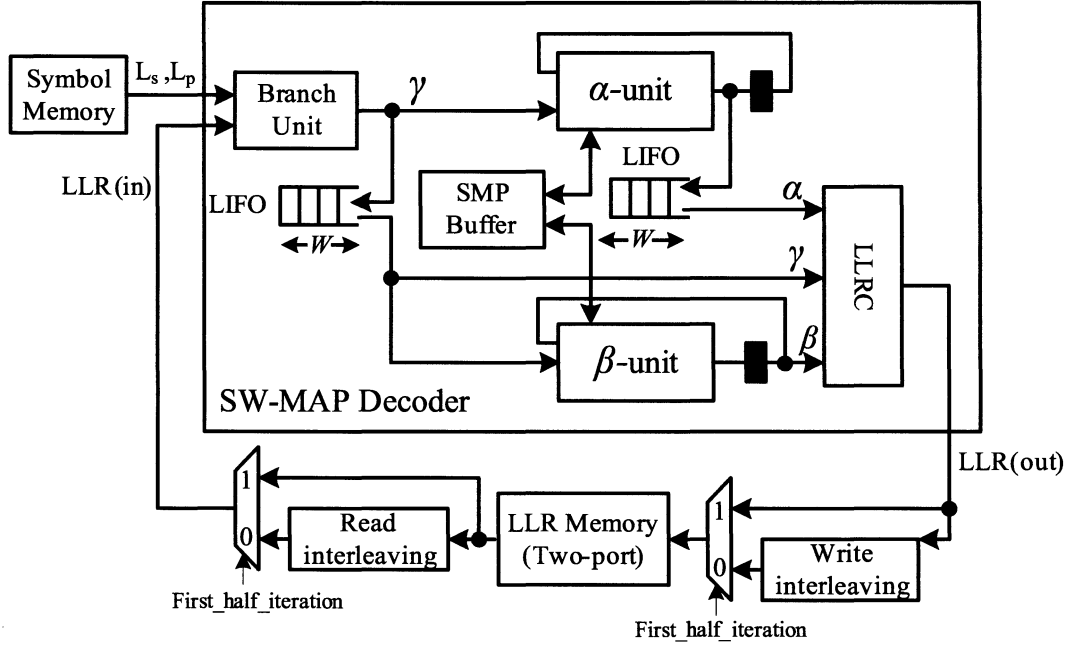


Figure 4.7 : SW-MAP decoder architecture.

shows an example of the addressing scheme for $W = 4$ and $x_0 = 0$. Figure 4.8(b) shows a hardware architecture for generating interleaving read/write addresses by using one forward QPP generator (cf. Figure 4.3) and one last-in first-out (LIFO) buffer.

When the sliding window length is large, using a LIFO can be costly. We will now propose another method to generate the interleaving write addresses. As depicted in Figure 4.9(b), a forward QPP address generator and a backward QPP address generator are used to recursively generate the read addresses $f(x)$ and write address $f(y)$, respectively. The initial values $f(x_0)$ and $g(x_0)$ for the forward QPP generator need to be pre-computed. However, the initial values for the backward QPP address

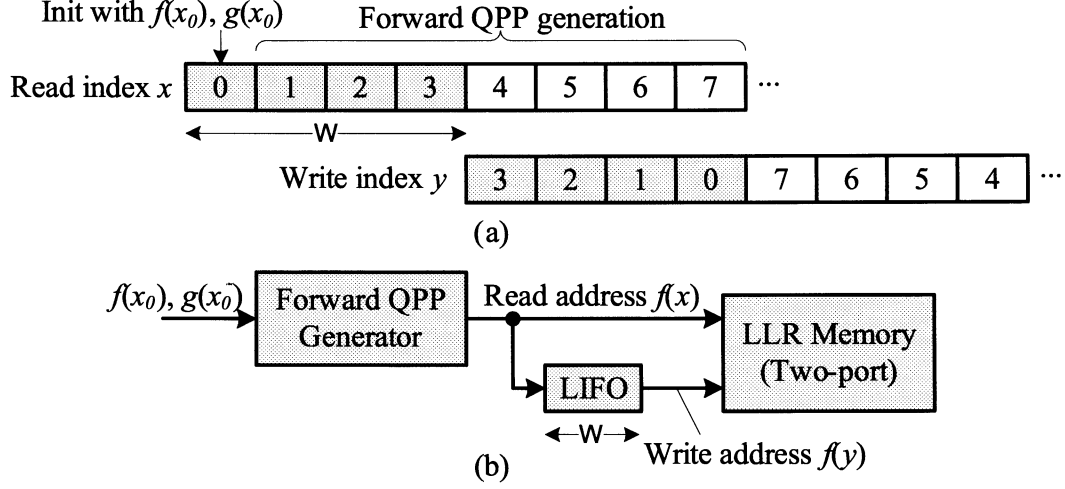


Figure 4.8 : (a) An example of the interleaver addressing scheme for the SW-MAP decoder, where $W = 4$, $x_0 = 0$. (b) Architecture for generating QPP interleaving read/write addresses.

generator are obtained from (synchronized with) the forward QPP address generator every W cycles and then a backward recursion is performed on the next $W - 1$ cycles to generate the next $W - 1$ write address. Figure 4.9(a) gives an example of this algorithm for $W = 4$ and $x_0 = 0$.

4.3.2 QPP Address Generator for Radix-4 SW-MAP Decoder

Radix-4 MAP decoding [52, 104] is a commonly used technique to achieve a higher trellis processing speed. For binary Turbo codes, eg. LTE Turbo codes, the trellis cycles can be reduced 50% by doing Radix-4 processing. In the Radix-4 processing, during the second half iteration two LLRs for information bit vector $\{u_x, u_{x+1}\}$ are needed to be fetched/written from/to the LLR memory at addresses $f(x)$ and $f(x+1)$. Thus, two read and two write interleaving addresses need to be generated in each clock

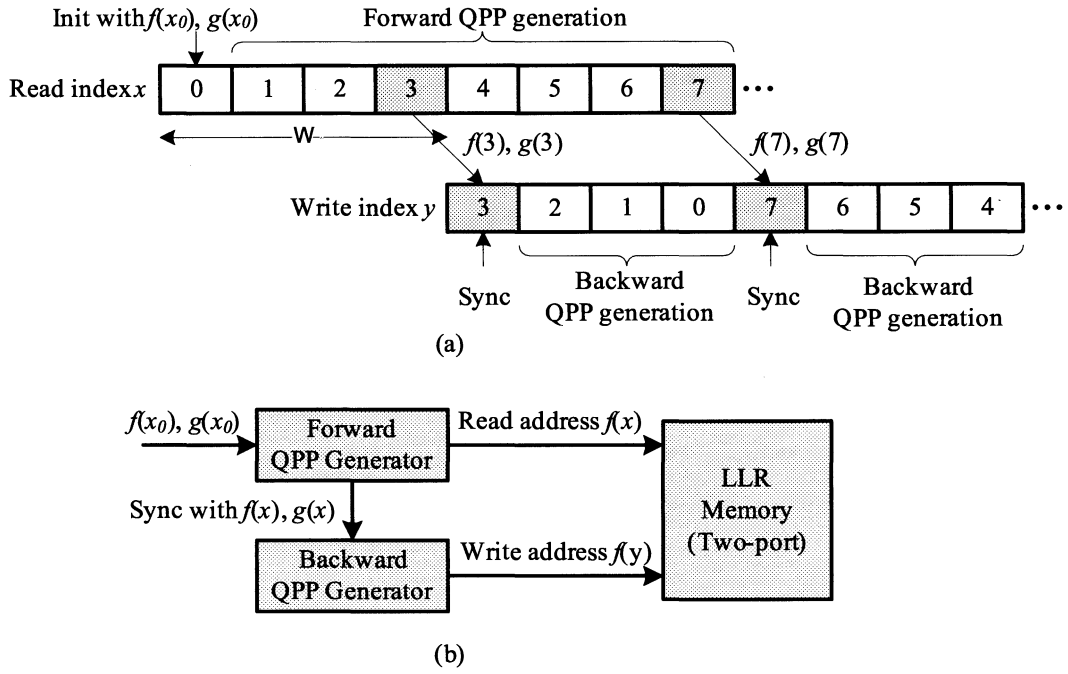


Figure 4.9 : (a) An example of the forward/backward data flow in SW-MAP algorithm, where $W = 4$. (b) A hardware architecture to generate interleaving read and write addresses for SW-MAP decoder.

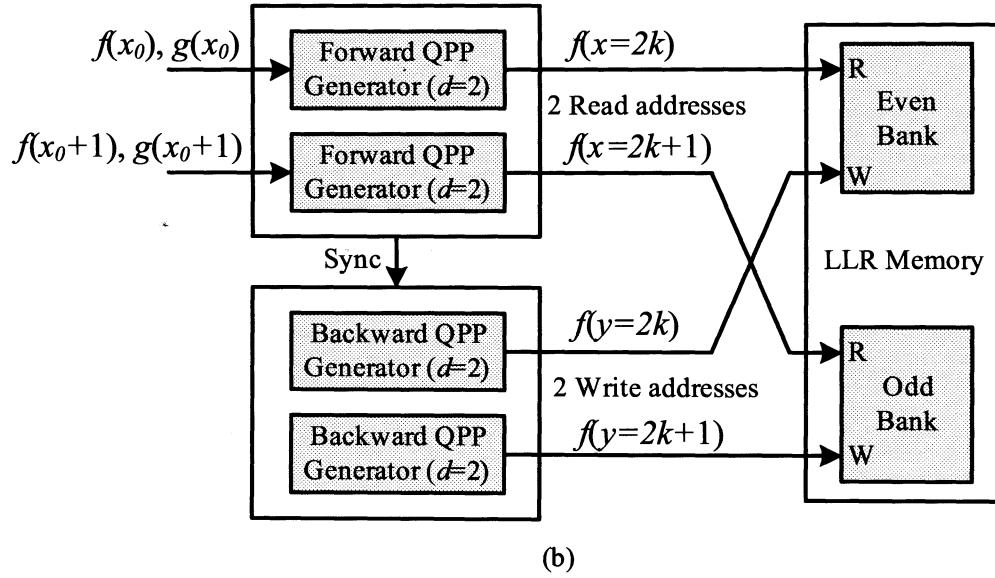
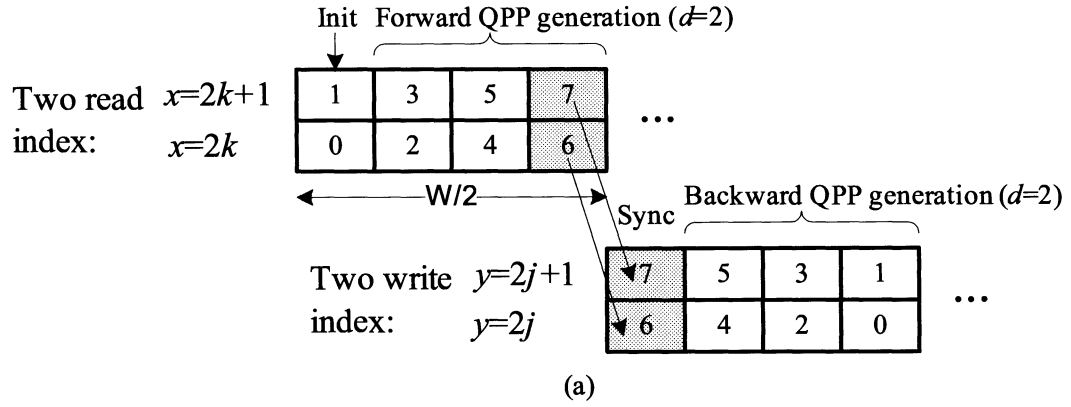


Figure 4.10 : (a) An example of the forward/backward data flow in Radix-4 SW-MAP algorithm, where $W = 4$. (b) A hardware architecture to generate read/write interleaving addresses for the Radix-4 SW-MAP decoder.

cycle. Figure 4.10(a) shows an example of the read/write addressing scheme where a sequence is partitioned into even and odd sub-sequences. Figure 4.10(b) shows a hardware architecture to generate the interleaving read and write addresses for the Radix-4 SW-MAP decoder. Two forward QPP address generators (with step $d = 2$) are used to generate the interleaving read addresses, and two backward QPP address generators (with step $d = 2$) are used to generate the interleaving write addresses. Based on the QPP algebraic property 1, the LLR memory can be partitioned into even and odd indexed banks to avoid collisions.

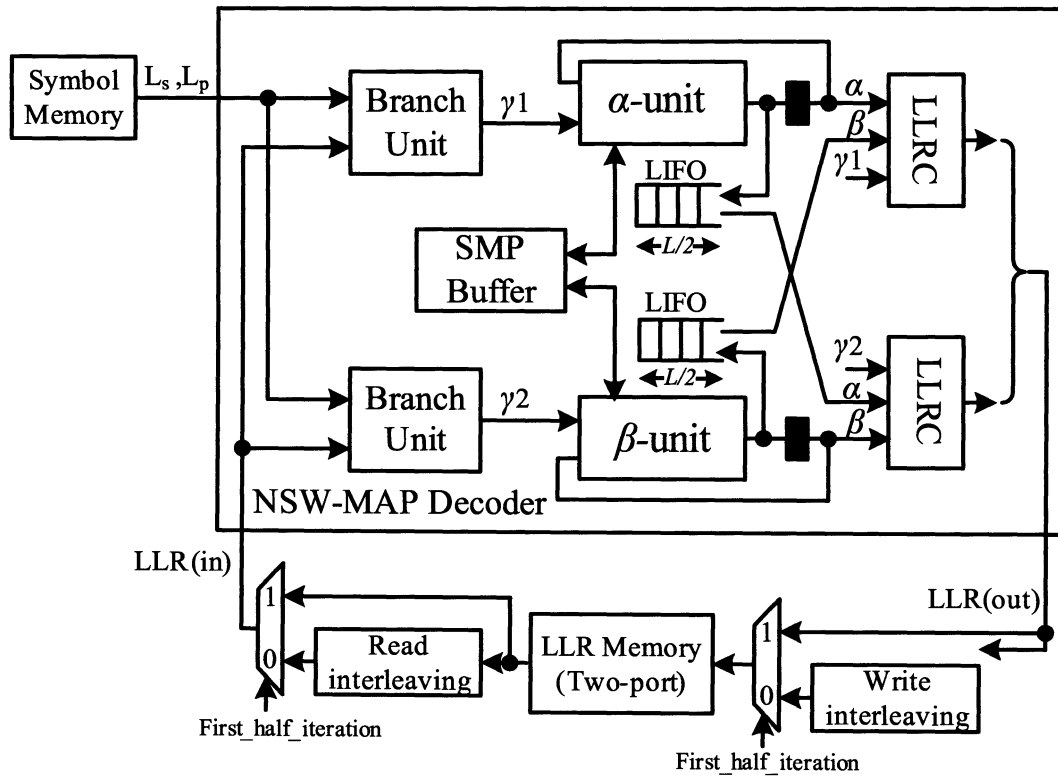


Figure 4.11 : NSW-MAP decoder architecture.

4.3.3 QPP Address Generator for NSW-MAP Decoder

In the NSW algorithm, forward and backward recursions are performed simultaneously by processing data from both ends of the sub-trellis. After the middle point, soft LLRs are calculated in both forward and backward directions. Figure 4.11 shows the NSW-MAP decoder architecture. Note that the NSW-MAP decoder requires two branch metric calculation units and two LLR calculation (LLRC) units because of the double-direction data processing. Figure 4.12(a) shows the forward/backward data flow in the NSW-MAP decoding process. Because both the forward and the backward processes need to access memory, we propose to use a two phase memory accessing scheme to support double-direction data processing. As shown in Figure 4.12(b), in phase 0, the forward MAP process is allowed to read two data at addresses $f(x)$ and $f(x + 1)$ from the LLR memory. In the next clock cycle (phase 1), the backward MAP process is allowed to read two data at addresses $f(y)$ and $f(y - 1)$ from the LLR memory. And then this process repeats. For the write operation, it is the same as the read operation. Also, the write address is just a delayed version of the read address. The number of delay cycles depends on the pipeline delays in the LLRC unit in the MAP decoder which is typically several clock cycles. Figure 4.12(c) shows a hardware architecture to implement this two-phase memory accessing algorithm, where the LLR memory is partitioned into even and odd indexed banks to avoid collisions. Each bank is a two-port memory module.

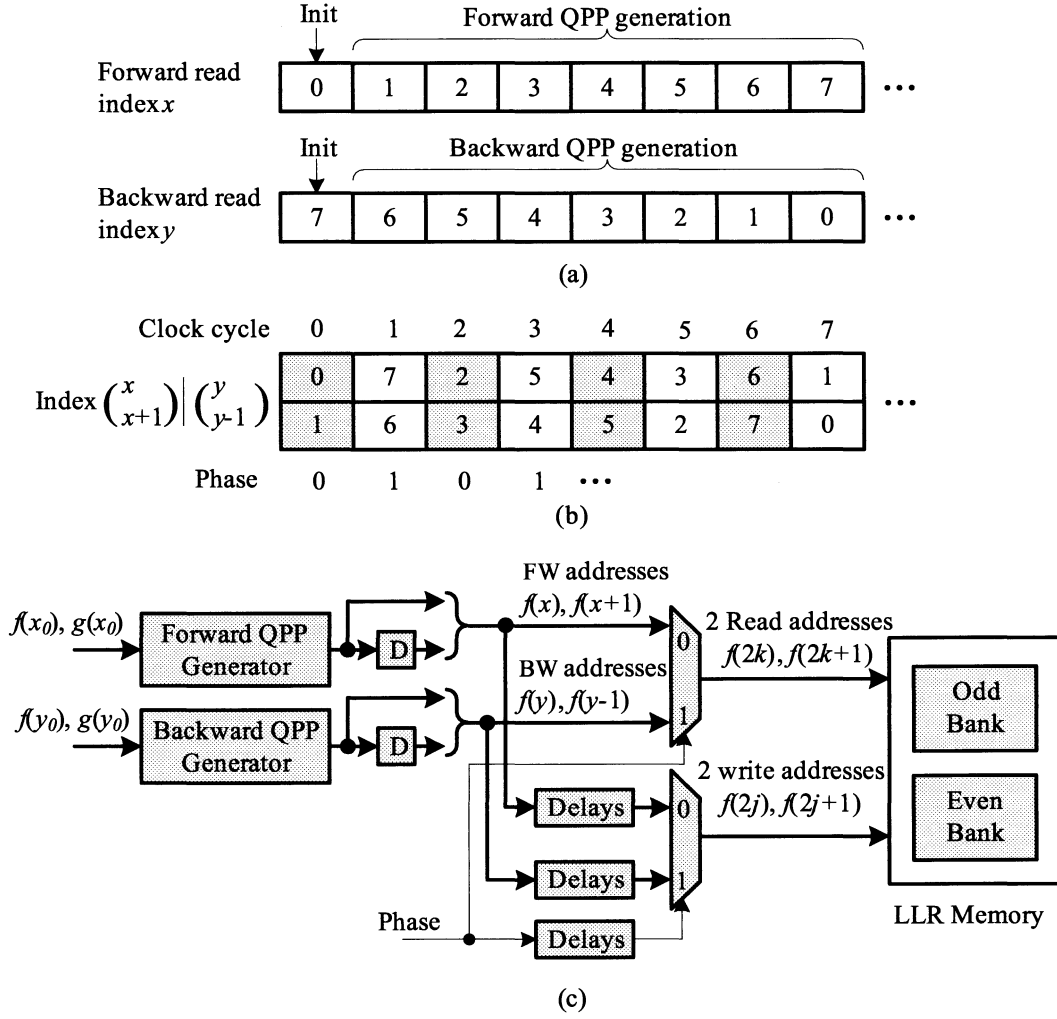


Figure 4.12 : (a) Forward/backward data flow in the NSW-MAP decoding process. (b) Two-phase memory accessing scheme. (c) A hardware architecture for generating interleaving addresses for the NSW-MAP decoder.

4.3.4 QPP Address Generator for Radix-4 NSW-MAP Decoder

The two-phase memory accessing scheme shown in Figure 4.12(b) can be extended to support Radix-4 NSW-MAP decoding as well, where four data at addresses $f(x)$, $f(x+1)$, $f(x+2)$, and $f(x+3)$ are needed to be generated in each clock cycle. Based on the QPP algebraic property 2 that the four consecutive interleaving addresses taking modulo 4 will lead to unique values, so the memory can be partitioned into four banks to allow four concurrent memory accesses in each clock cycle without any collisions. Figure 4.13 shows a hardware architecture for generating interleaving addresses for the Radix-4 NSW-MAP decoder.

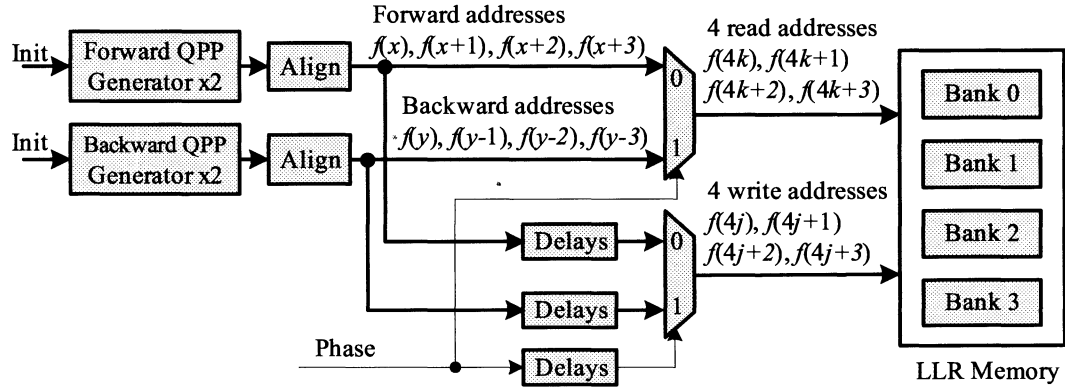


Figure 4.13 : A hardware architecture for generating interleaving addresses for the Radix-4 NSW-MAP decoder.

4.3.5 MAP Decoder Comparison

Table 4.2 compares the resource usage and decoding latency for a SW-MAP decoder and a NSW-MAP decoder, in which W is the sliding window length in the SW

algorithm, L is the segment length $L = N/P$, B_α and B_γ are the total bit widths for the α state metrics (8 states in total) and the γ branch metrics, respectively.

Table 4.2 : MAP decoder architecture comparison.

	SW-MAP	NSW-MAP
α unit	1	1
β unit	1	1
Branch unit	1	2
LLRC	1	2
QPP address generator	2	2
State-buffer (bit)	$B_\alpha \times W$	$B_\alpha \times L$
γ -buffer (bit)	$B_\gamma \times W$	0
SMP-buffer (bit)	$B_\alpha \times 2L/W$	$B_\alpha \times 4$
Processing time (cycles)	$W + L$	L

The sub-block size W depends on the parallelism level P in a parallel Turbo decoder architecture where multiple MAP decoders are employed. Figure 4.14 illustrates the two parallel decoding algorithms based on the SW-MAP decoder and the NSW-MAP decoder. In this particular example, $P = 4$ number of MAP decoders are used.

To compare the area for these two types of MAP decoder architectures, we have synthesized them in a TSMC 65-nm CMOS technology for a 400 MHz clock frequency. The fixed point word lengths for the channel LLRs, extrinsic LLRs, and state metrics are 6, 7, and 10 respectively [12]. For the SW-MAP architecture, the sliding window

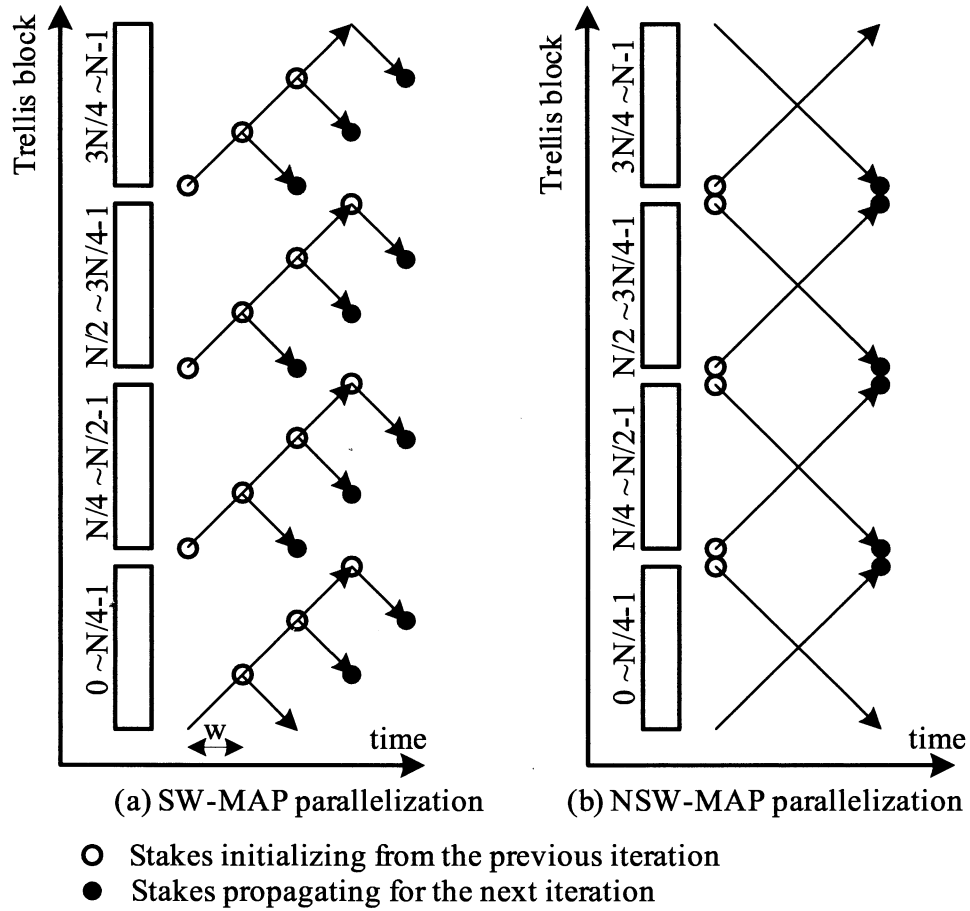


Figure 4.14 : An example of a multi-MAP parallel decoding approach with $P = 4$. (a) Parallel SW-MAP algorithm with state metric propagation. (b) Parallel NSW-MAP algorithm with state metric propagation.

length W is assumed to be 64. Consider decoding of a segment of a code block where the code length is $N = 6144$ and the segment length is $L = N/P$, Figure 4.15 shows the area cost for these two types of MAP decoders. As can be seen, as the decoder parallelism P increases, the area cost of the NSW-MAP decoder reduces quickly and comes closer to the area cost of the SW-MAP decoder.

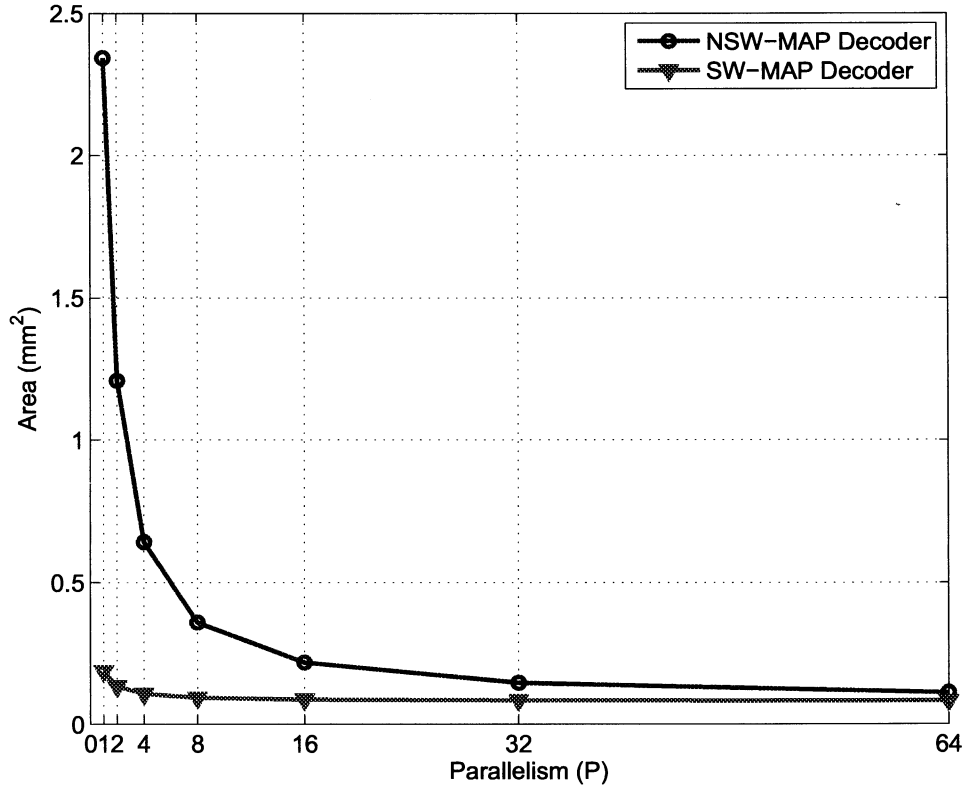


Figure 4.15 : Area of a NSW-MAP decoder and a SW-MAP decoder.

To compare the efficiency of these two architectures, we define an efficiency metric as **area** \times **time**, or AT, where area is one MAP decoder area and time is the processing time for a sub-trellis for a half Turbo iteration. Figure 4.16 plots the

AT complexities for different P , where the AT value is displayed on a logarithmic scale. Clearly, when the parallelism degree P is small, the NSW-MAP architecture has a higher AT complexity than the SW-MAP architecture because a large number of state metrics have to be buffered. On the other hand, as P increases, the NSW-MAP architecture will become more efficient due to the fact that the double-flow NSW-MAP decoding has no sliding window overhead, whereas the single-flow SW-MAP decoding has a sliding window overhead of $\frac{W}{(N/P+W)}$. As a design tradeoff, we adopted the SW-MAP architecture in our final hardware implementation to save area while still achieving 1Gbps throughput.

Figure 4.17 compares the AT complexities of a Radix-4 SW-MAP decoder and a Radix-4 NSW-MAP decoder for a 250 MHz clock frequency. One observation is that the Radix-4 transform can effectively reduce the AT complexity of the NSW-MAP decoder when P is small. However, Radix-4 transform will not necessarily reduce the AT complexity of the SW-MAP decoder. This is due to the fact that the Radix-2 decoder can run at a faster clock frequency, and has a lower complexity than the Radix-4 decoder (assuming full LogMAP implementation). We will compare the Radix-2 and the Radix-4 architectures in more detail in the next section.

4.4 Top Level Parallel Turbo Decoder Architecture

Decoder parallelism is necessary to achieve the LTE/LTE-Advance high throughput requirement which is up to 1 Gbps. In order to increase the throughput by a factor

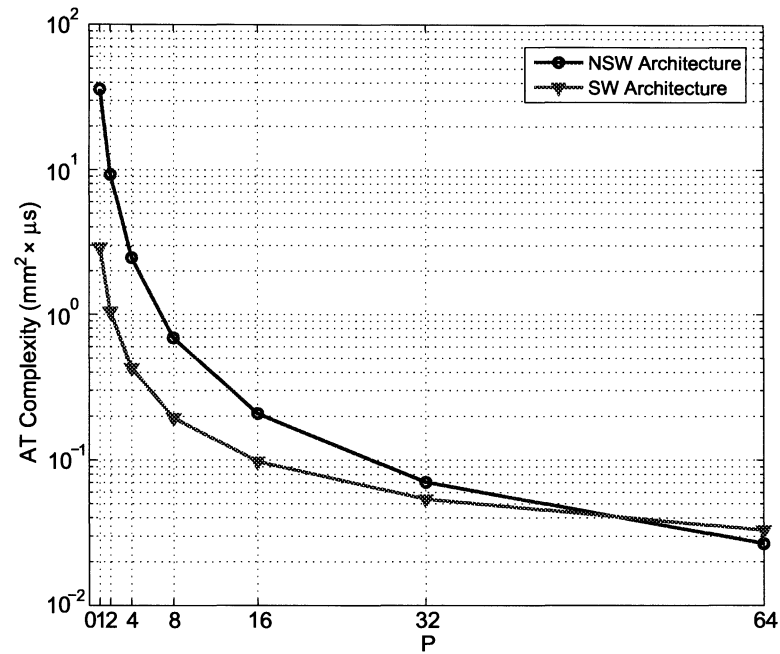


Figure 4.16 : AT complexity of a SW-MAP decoder and a NSW-MAP decoder.

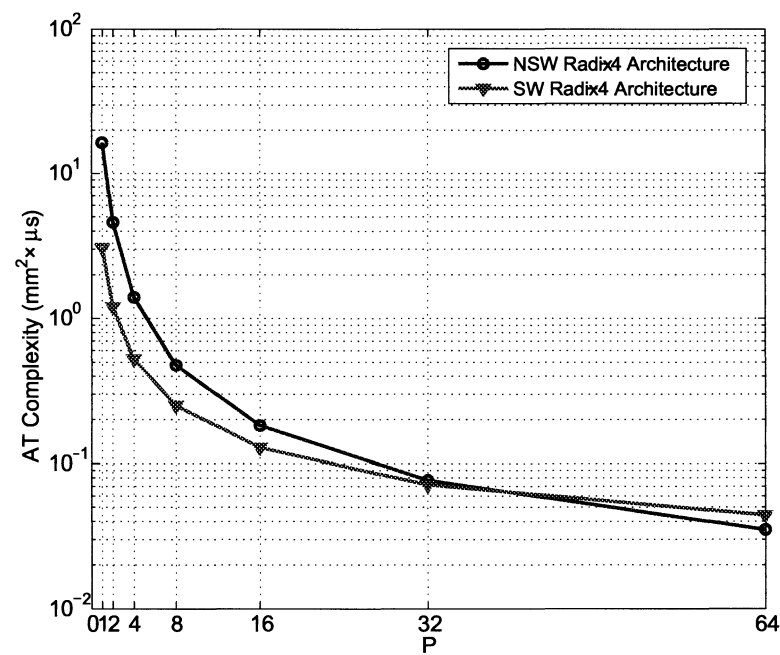


Figure 4.17 : AT complexity of a Radix-4 SW-MAP decoder and a Radix-4 NSW-MAP decoder.

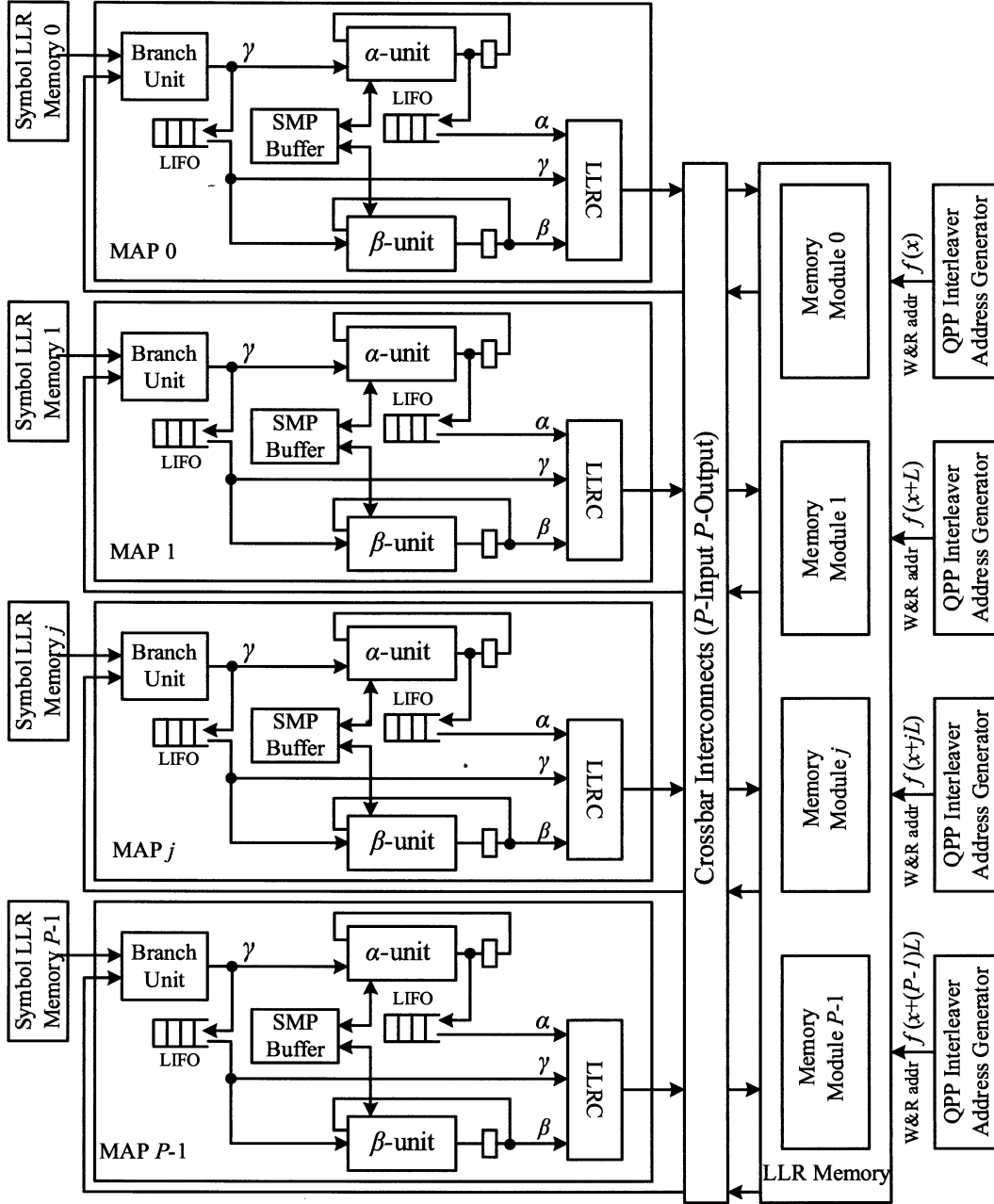


Figure 4.18 : The proposed parallel decoder architecture with P SW-MAP decoders. P memories are used to support contention-free memory accessing. Crossbar interconnects are used to permute the memory read/write data.

of P , an information block can be divided into P segments with equal length L and then each segment is processed independently by a dedicated MAP decoder [111, 112, 113, 114, 103, 115, 116, 117, 12, 53, 58]. In this scheme, each of the P MAP cores processes the data sequentially and fetches/writes the data simultaneously always at the same offset x to each segment. The interleaver structure in the current and previous 3G standards do not have a parallel structure which makes it difficult to realize the parallelization of the MAP decoders. Expensive write buffers have to be used to reduce the memory collision caused by the interleaver [93, 118]. However, when the parallelism degree increases, the collisions can not be effectively resolved by using write buffers. The LTE QPP interleaver, however, has an inherent parallel structure that supports contention-free memory accesses which result in a large design space for the selection of appropriate levels of decoder parallelism.

In this section, we will present a highly-parallel Turbo decoder architecture based on the QPP conflict-free interleaver and give an analysis of the complexity and the throughput. Figure 4.18 shows a hardware architecture for implementing the proposed parallel SW-MAP algorithm. In this architecture, P sets of QPP interleavers are used to generate the interleaving addresses $f(x)$, $f(x+L)$, ..., and $f(x+(P-1)L)$ concurrently, where L is the segment length $L = N/P$. Based on the QPP contention-free property, these P addresses will be mapped to different memory modules 0 to $P-1$ without any collisions. Thus, no write buffers are required. A crossbar network is used to permute the data between the MAP decoders and the memory modules.

Furthermore, based on the QPP interleaver algebraic property 3, this architecture can be modified to support the Radix-4 SW and NSW MAP decoding algorithms by setting the following constraints. To support the Radix-4 SW-MAP decoding, L needs to be divisible by 2, and each memory module needs to be partitioned into even and odd indexed banks. To support the Radix-4 NSW-MAP decoding, L needs to be divisible by 4, and each memory module needs to be partitioned into four banks.

4.4.1 Throughput-Area Tradeoff Analysis

High throughput is achieved by using multiple MAP decoders and multiple memory modules/banks. In this section, we will analyze the impact of parallelism on throughput and area. The maximum throughput is measured as:

$$\begin{aligned} \text{SW Throughput} &= \frac{N}{\text{Decoding time}} \approx \frac{N \cdot f}{I \cdot (\tilde{N}/P + \tilde{W})} \\ \text{NSW Throughput} &= \frac{N}{\text{Decoding time}} \approx \frac{N \cdot f}{I \cdot (\tilde{N}/P)}, \end{aligned}$$

where $\tilde{N} = N$, $\tilde{W} = W$ in the case of Radix-2 decoding, and $\tilde{N} = N/2$, $\tilde{W} = W/2$ in the case of Radix-4 decoding. I is the total number of half iterations performed by the Turbo decoder. f is the operating clock frequency.

To analyze the area and throughput performance for different QPP parallelism degrees, we describe a Radix-2 and a Radix-4 SW parallel Turbo decoder in Verilog HDL and synthesize these decoders for a 65 nm CMOS technology using Synopsys Design Compiler. The tradeoff analysis result is given in Figures 4.19 and 4.19 which plots the area and the throughput for different parallelism degrees and clock rates. As

can be seen, a 1 Gbps throughput is achievable with 64 Radix-2 MAP decoder cores running at a 310MHz clock frequency or 32 Radix-4 MAP decoder cores running at a 250MHz clock frequency.

For a parallel Turbo decoder which consists of multiple MAP units, the MAP units tend to dominate the silicon area especially when the parallelism is high. From Figures 4.19 and 4.20, we can see that given the same throughput target, the Radix-2 architecture provides a lower area cost than the Radix-4 architecture for most of the cases and especially when P is large. This is mainly due to the fact that the Radix-2 MAP unit can run at a faster clock frequency, and has a lower complexity than the Radix-4 MAP unit (assuming full LogMAP implementation). However, it should be noted that the Radix-2 decoder may need a higher partitioning of the code block than the Radix-4 decoder to achieve the same throughput target. As a design tradeoff, we adopted the Radix-2 architecture in our final hardware implementation to save area while still meeting the 1 Gbps throughput target.

4.5 Summary

We have presented a highly-parallel Turbo decoder architecture for LTE-Advance system. By utilizing the new contention-free interleaver, we designed a 64-MAP parallel decoder to achieve 1+ Gbps data rate. Compared to the existing 3G or 4G Turbo decoders, the proposed Turbo decoder has a significant throughput advantage while still maintaining low area cost and low power consumption. In Chapter 6, we

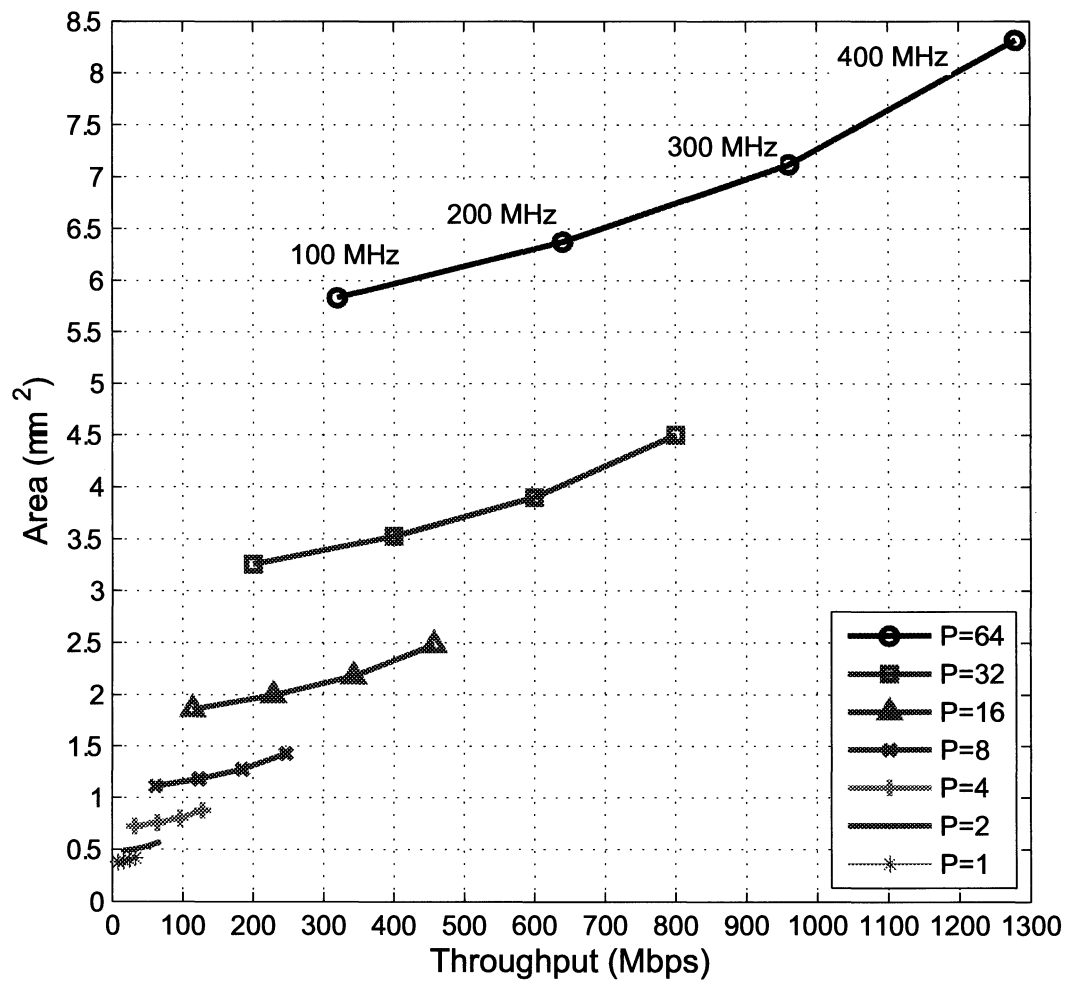


Figure 4.19 : Area-throughput tradeoff analysis for Radix-2 Turbo decoder

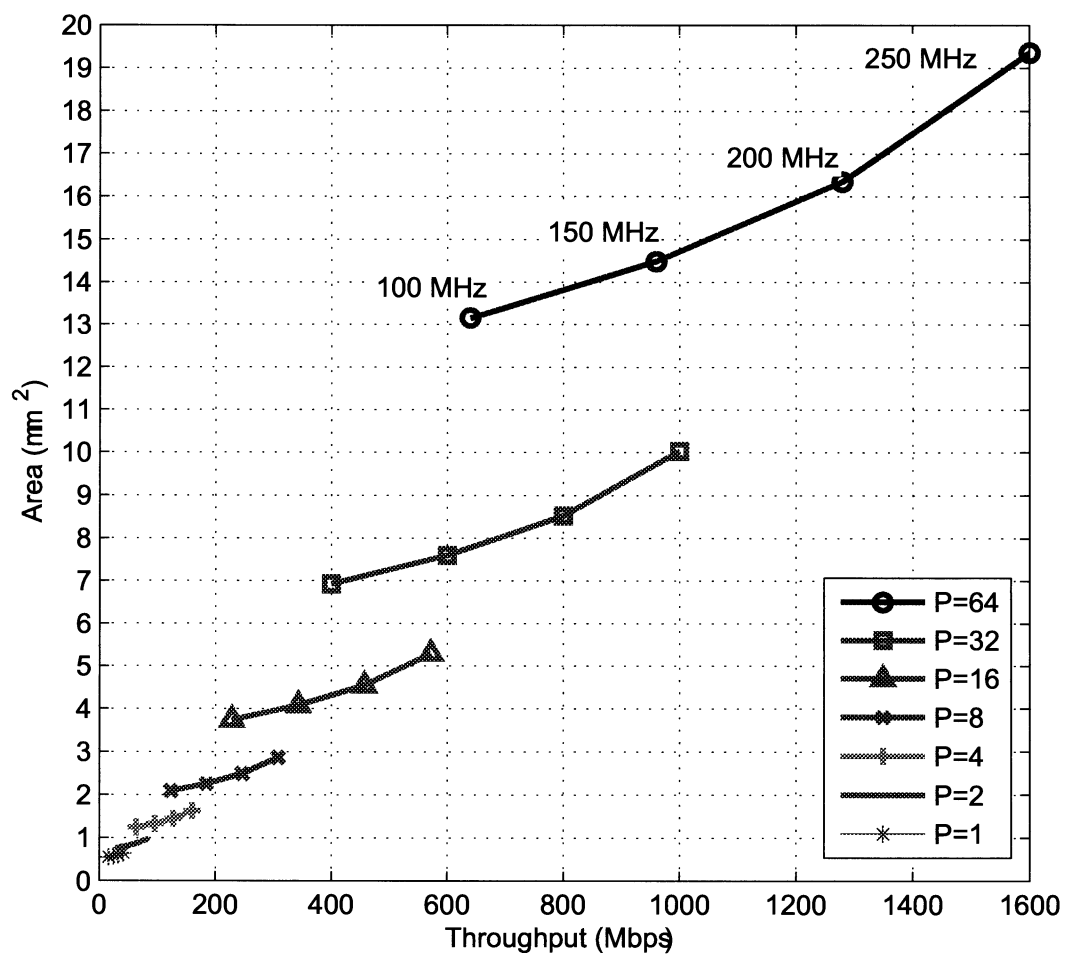


Figure 4.20 : Area-throughput tradeoff analysis for Radix-4 Turbo decoder.

will present the ASIC implementation results for the proposed Turbo decoder in more details. To support iterative detection and decoding scheme, this Turbo decoder can be configured to output soft LLR values to the detector.

Chapter 5

High-Throughput LDPC Decoder Architecture

LDPC codes have inherent large parallelism that can be exploited to design a high-speed decoder. In theory, a random LDPC code with infinite block size will achieve near-capacity performance. However, it is very complex to implement such a decoder because of the random parity check matrix. To reduce implementation complexity while still maintaining good error protection capability, new wireless standards are adopting structured quasi-cyclic LDPC (QC-LDPC) codes. These structured QC-LDPC codes typically have a block size of several thousands bits and can be either regular codes and irregular codes. If the parity check matrix of a LDPC code has the same row and column degree, this LDPC code is called a regular LDPC code. Otherwise, it is an irregular LDPC code.

Partial-parallel architectures are often used for the decoding of these structured QC-LDPC codes. The main challenge of the partial-parallel architecture is to develop a flexible decoder architecture to support multiple codes. The existing LDPC decoders are developed mostly for a particular standard which lacks the flexibility to be reconfigured to support multiple standards. In this chapter, we describe high-throughput low-density parity-check (LDPC) decoder architectures that support variable block sizes and multiple code rates. Various techniques are used to reduce the

implementation complexity of the LDPC decoders. We first present a Min-sum algorithm based LDPC decoder. Next, we present a more powerful Log-MAP algorithm based LDPC decoder. To achieve multi-Gbps decoding throughput, we propose a multi-layer parallel decoder architecture. Furthermore, we propose a flexible decoder architecture that can support both LDPC codes and Turbo codes with a low hardware overhead.

5.1 Structured QC-LDPC Codes

In chapter 2, we have introduced the general LDPC codes. Almost all the practical wireless systems currently use the QC-LDPC codes. In this chapter, we mainly focus on the decoder design for the structured QC-LDPC codes. As shown in Fig. 5.1(a)(b), for a QC-LDPC code, the parity check matrix (PCM) is constructed from an $M \times N$ seed matrix by replacing each '1' in the seed matrix with a $Z \times Z$ cyclically shifted identity sub-matrix, where Z is an expansion factor. A corresponding Tanner factor graph representation of this $MZ \times NZ$ generated PCM is shown in Fig. 5.1(c). It divides the variable nodes and the check nodes into clusters of size Z such that if there exists an edge between variable and check clusters, then it means Z variable nodes connect to Z check nodes via a permutation (cyclic shift) network.

As an example, Fig. 5.2 shows the parity check matrix for the block length 1944 bits, code rate 1/2, sub-matrix size $Z = 81$, IEEE 802.11n LDPC code. In this matrix representation, each square box with a label I_x represents an 81×81 cyclicly-shifted

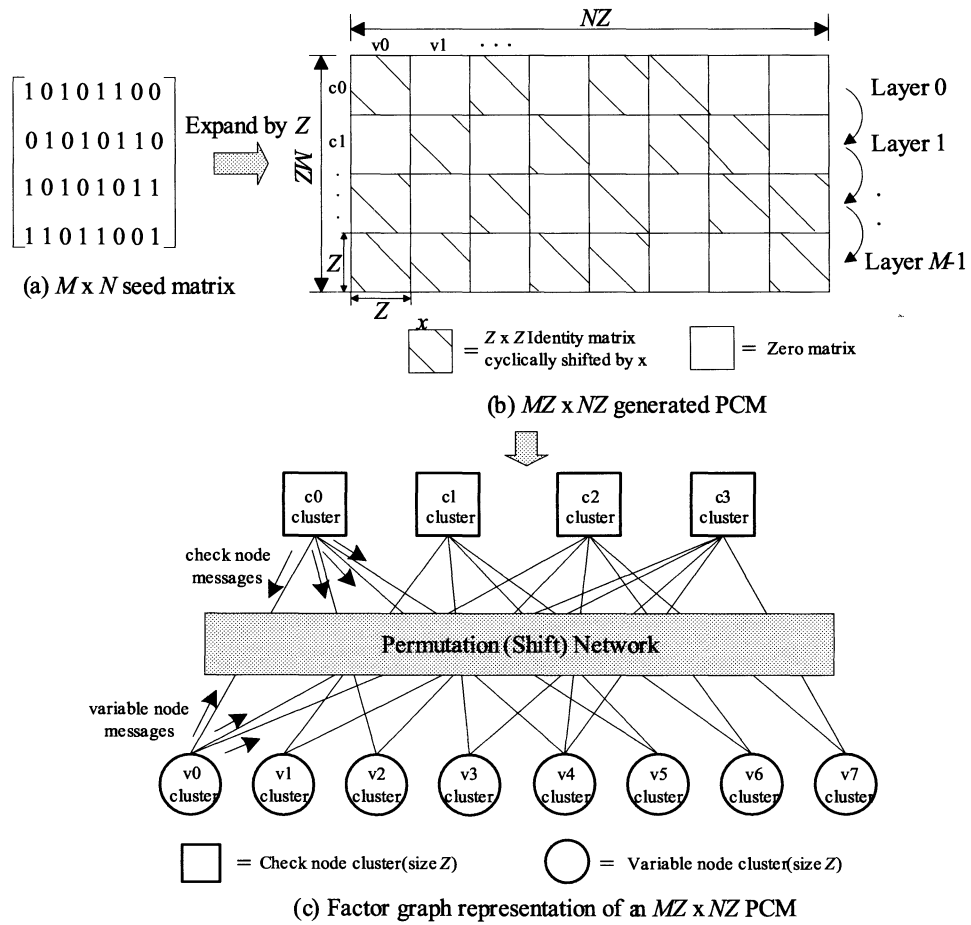


Figure 5.1 : Parity check matrix and its factor graph representation

n is defined as:

$$L_n = \log \frac{Pr(n=0)}{Pr(n=1)}, \quad (5.1)$$

where L_n is initialized to be the channel input LLR. The check node message from check node m to variable node n is denoted as $R_{m,n}$. The variable message from variable node n to check node m is denoted as $Q_{m,n}$. The conventional layered algorithm, or single-layer algorithm, assumes that the rows are grouped into layers where the parity check matrix for this layer has at most a column-weight of one. The single-layer algorithm only handles one layer at a time, i.e. the maximum row parallelism is limited to the sub-matrix size Z . Each layer is processed as a unit, one layer after another. For each non-zero column n inside the current layer, variable node messages $Q_{m,n}$ that correspond to a row m are formed by subtracting the check node message $R_{m,n}$ from the APP LLR message L_n :

$$Q_{m,n} = L_n - R_{m,n}. \quad (5.2)$$

For each row m , the new check node messages $R'_{m,n}$, corresponding to all variable nodes j that participate in this parity-check equation, are computed using the belief propagation algorithm. In this work, we use the scaled min-sum approximation algorithm (with scaling factor of S) to compute the R value:

$$R'_{m,n} = \prod_{j \in \mathcal{N}_m \setminus n} \text{sign}(Q_{m,j}) \cdot \Psi \left(\sum_{j \in \mathcal{N}_m \setminus n} \Psi(Q_{m,j}) \right), \quad (5.3)$$

where \mathcal{N}_m is the set of variable nodes that are connected to check node m , and $\mathcal{N}_m \setminus n$ is the set \mathcal{N}_m with variable node n excluded. The non-linear function $\Psi(x)$ is defined

as:

$$\Psi(x) = -\log \left[\tanh \left(\frac{|x|}{2} \right) \right]. \quad (5.4)$$

To reduce implementation complexity, the min-sum algorithm [63, 64] can be used to approximate the non-linear function $\Psi(x)$. By applying the scaled min-sum algorithm with a scaling factor of S , equation (5.3) is changed to:

$$R'_{m,n} \approx S \cdot \prod_{j \in \mathcal{N}_m \setminus n} \text{sign}(Q_{m,j}) \cdot \min_{j \in \mathcal{N}_m \setminus n} |Q_{m,j}|, \quad (5.5)$$

where \mathcal{N}_m is the set of variable nodes that are connected to check node m , and $\mathcal{N}_m \setminus n$ is the set \mathcal{N}_m with variable node n excluded. After the check nodes messages are computed, the new APP LLR messages L'_n are updated as:

$$L'_n = L_n + R'_{m,n} - R_{m,n}. \quad (5.6)$$

The layered decoding algorithm is often used to decode the structured QC-LDPC codes. In chapter 2, we have introduced the layer decoding algorithm in detail. We summarize the layered decoding algorithm in Algorithm 3.

5.3 Block-Serial Scheduling Algorithm

To implement Algorithm 3 in hardware, we propose a block-serial (BS) scheduling algorithm as shown in Fig. 5.3. In this algorithm, one full iteration is divided into M sub iterations. A processing element (PE) is applied to each layer in sequence. Each $Z \times Z$ sub-matrix is treated as a macro within which all the involved parity checks

Algorithm 3 Layered belief propagation algorithm

Initialization:
 $\forall(m, n)$ with $H(m, n) = 1$, set $R_{mn} = 0$, $L_n = \frac{2y_n}{\sigma^2}$
for iteration $i = 1$ to I **do**

 for layer $l = 1$ to L **do**

 1) **Read:**
 $\forall(m, n)$ with $H^l(m, n) = 1$:

 Read L_n and R_{mn} from memory

 2) **Decode:**

$$Q_{mn} = L_n - R_{mn}$$

$$R_{mn}^{new} = \prod_{j \in \mathcal{N}_m \setminus n} \text{sign}(Q_{mj}) \Psi \left(\sum_{j \in \mathcal{N}_m \setminus n} \Psi(Q_{mj}) \right)$$

$$L_n^{new} = Q_{mn} + R_{mn}^{new}$$

 3) **Write back:**

 Write L_n^{new} and R_{mn}^{new} back to memory

 end for
end for
Decision making: $\hat{x}_n = \text{sign}(L_n)$

are processed in parallel using Z number of PEs. Each PE is independent from all others since there is no data dependence between adjacent check rows.

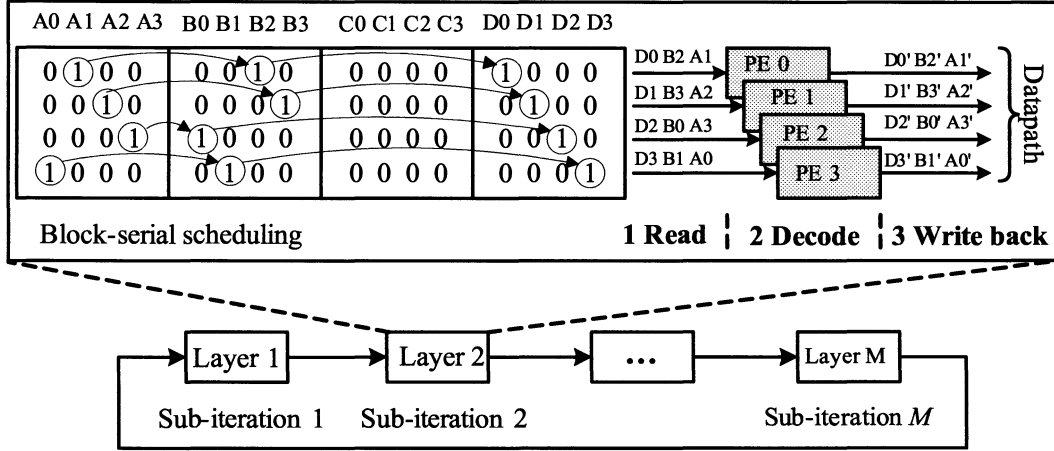


Figure 5.3 : Block-serial (BS) scheduling algorithm

5.4 Min-sum LDPC Decoder Architecture

Fig. 5.4 shows the block diagram of the decoder architecture based on the layered min-sum decoding algorithm. In each sub-iteration, a cluster of APP messages and check messages are fetched from APP and Check memory, and then the APP messages are passed through a flexible permuter to be routed to the correct Processing Engines (PEs) for updating new APP messages and check messages. The PEs are the central processing units of the architecture that are responsible for updating the check node and variable node messages. The number of PEs determines the parallelism factor of the design. For a certain block-size code, only Z PEs are working while the rest are in a power saving mode. As shown in Fig. 5.5, the PE inputs wr elements of L_n

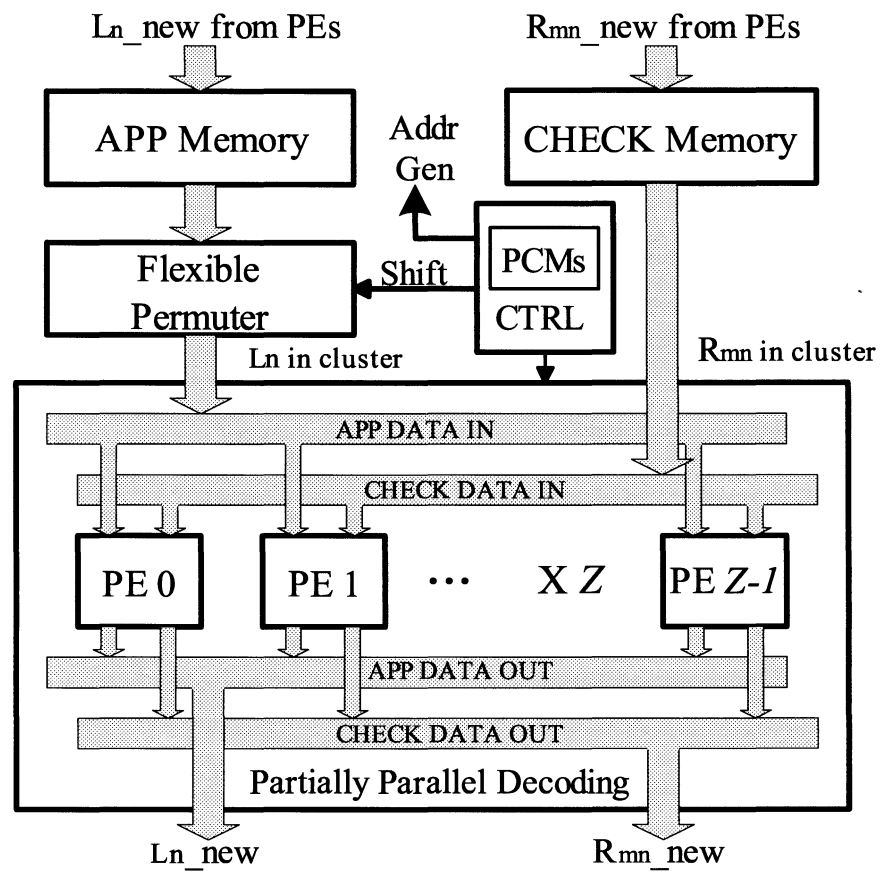


Figure 5.4 : Top level min-sum LDPC decoder architecture

and R_{mn} , where w_r is the number of nonzero values in each row of the PCM. Q_{mn} is calculated based on (5.2). The sign and magnitude of Q_{mn} are processed based on (5.5) to generate new R_{mn} . Then the Q_{mn} are added to the R_{mn} to generate new L_n (w_r of them) based on (5.6). The outputs (L_n and R_{mn}) of all the Z PEs are concatenated and stored in one address of the APP and Check memories. For each layer's sub-iteration, it takes about $2w_r$ clock cycles to process, so the decoding throughput is:

$$Throughput \approx \frac{N \times Z \times Rate \times fclk_{max}}{2 \times E \times iterations}$$

where $Rate$ is the code rate and E is the total number of edges between all variable nodes and check nodes in the seed matrix. Clearly, the throughput would be linearly proportional to the expansion factor Z for a given seed matrix.

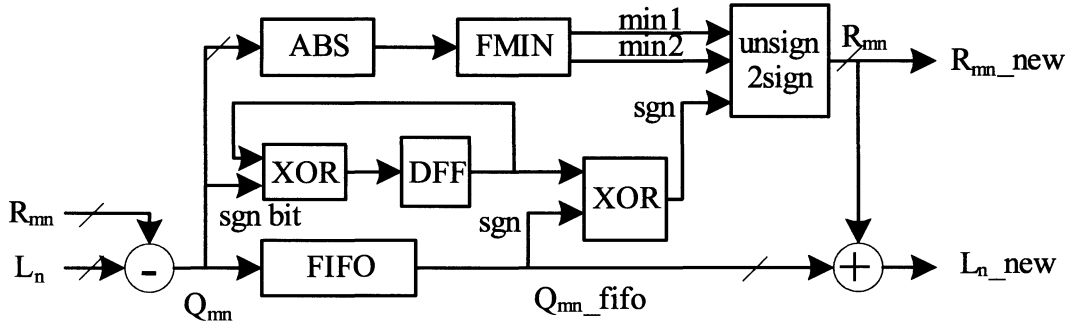


Figure 5.5 : Processing Engine (PE)

5.4.1 Flexible Permuter Design

One of the main challenges of the LDPC decoder architecture is the permuter design that is responsible for routing the messages between variable nodes and check nodes. However for QC-LDPC codes, the permuter is just a barrel shifter network (size- Z) for cyclically shifting the node messages to the correct PEs. Fig. 5.6 gives an example of a size-4 barrel shifter network. The hardware design complexity of this type of network is $O(Z \lceil \log_2 Z \rceil)$ as compared to $O(Z^2)$ for the directly connected network. For large size Z (e.g. 128), the barrel shifter network needs to be partitioned into multiple pipeline stages for high speed VLSI implementation.

Traditionally a de-permuter would be needed to permute the shuffled data back and save it to memory, which would occupy a significant portion of the chip area [80]. However, due to the cyclic shift property of the QC-LDPC codes, no de-permuter is needed. We can just store the shuffled data back to memory and for the next iteration we should then shift this "shuffled data" by an incremental value $\Delta = (\text{shift}_n - \text{shift}_{n-1}) \bmod Z$.

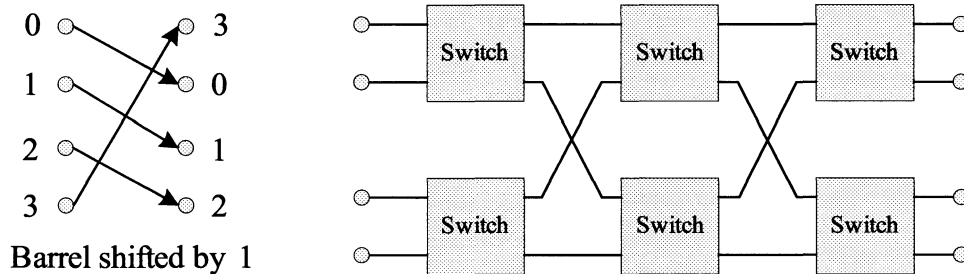


Figure 5.6 : A 4×4 Barrel shifter network

5.4.2 Pipelined Decoding for Higher Throughput

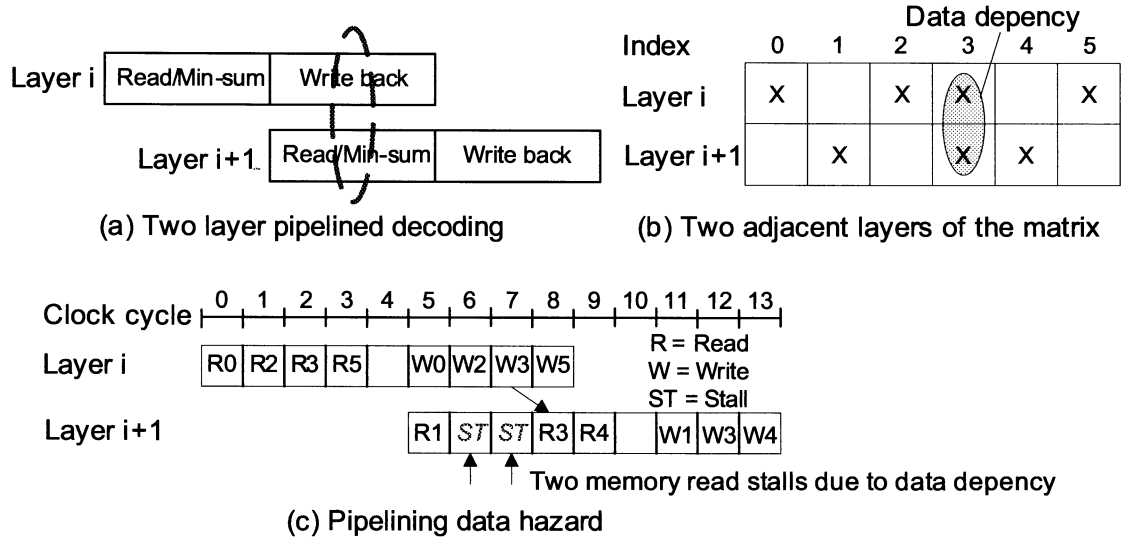


Figure 5.7 : Pipelined decoding

The decoding throughput can be further improved by overlapping the decoding of two layers using a pipelined method. The decoding of each layer of the parity check matrix is performed in two stages: 1) Memory read and min-sum calculation and 2) Memory write back. However, due to the possible data dependence between two consecutive layers (there is no data dependency inside each layer because the column weight is at most 1 in each layer), a pipelining data hazard might occur. Fig. 5.7 shows an example of pipelined decoding. In Fig. 5.7(c), at clock cycle 6, layer $(i + 1)$ is trying to access APP memory address 3 which will not be updated by layer i until clock cycle 7, hence two pipeline stalls need to be inserted. Moreover, a horizontal rescheduling algorithm can also be applied to help reduce pipeline stalls.

For example, in Fig. 5.7, layer $(i + 1)$'s reading can be rescheduled from the original sequence 1-3-4 to 1-4-3 to reduce pipeline stalls. This way, the decoding throughput will be increased to

$$\text{Pipelined Throughput} \approx \frac{N \times Z \times \text{Rate} \times f_{clk}}{E \times I},$$

where I is the number of iterations.

5.5 Log-MAP LDPC Decoder Architecture

5.5.1 Low-Complexity Implementation of The Log-MAP Algorithm

Conventionally, function $\Psi(x) = -\log(\tanh(|x/2|))$ is used for the decoding operations in Algorithm 3. However, the $\Psi(x)$ function is prone to quantization noise and can be numerically unstable [119]. Alternately, a different and numerically more robust way to compute the R_{mn} is shown as

$$R_{mn} = \sum_{j \in \mathcal{N}_m \setminus n} \boxplus Q_{mj} = \left(\sum_{j \in \mathcal{N}_m} \boxplus Q_{mj} \right) \boxminus Q_{mn}, \quad (5.7)$$

where the \boxplus and \boxminus operations are defined as $a \boxplus b \triangleq f(a, b) = \log \frac{1+e^a e^b}{e^a + e^b}$ and $a \boxminus b \triangleq g(a, b) = \log \frac{1-e^a e^b}{e^a - e^b}$ [120][121]. This computation method is especially suitable for the proposed BS scheduling algorithm in which the macro blocks are processed in sequential order. For hardware implementation, $f(\cdot)$ and $g(\cdot)$ functions can be

simplified to

$$\begin{aligned}
 f(a, b) &= \text{sign}(a) \text{sign}(b) \left(\min(|a|, |b|) + \right. \\
 &\quad \left. \log(1 + e^{-(|a|+|b|)}) - \log(1 + e^{-||a|-|b||}) \right), \\
 g(a, b) &= \text{sign}(a) \text{sign}(b) \left(\min(|a|, |b|) + \right. \\
 &\quad \left. \log(1 - e^{-(|a|+|b|)}) - \log(1 - e^{-||a|-|b||}) \right).
 \end{aligned} \tag{5.8}$$

In hardware, the non-linear correction terms $\log(1 + e^{-x})$ and $\log(1 - e^{-x})$ in (5.8) are approximated using low-complexity 3-bit lookup tables (LUTs) [121].

5.5.2 Radix-2 Log-MAP SISO Decoder

Fig. 5.8 shows the proposed soft-input soft-output (SISO) decoder architecture for generating R_{mn} . We refer to it as Radix-2 (R2) recursion architecture since only one element can be processed in one clock cycle. The R2-SISO core consists of one $f(\cdot)$ recursion unit followed by one $g(\cdot)$ unit. Note that the $g(\cdot)$ unit would have the same structure as the $f(\cdot)$ unit but with a different LUT.

Fig. 5.9 shows the decoding schedule for check row m . During the first d_m * cycles, the incoming variable messages Q_{mn} ($\forall n \in \mathcal{N}_m$) are fed to the decoder sequentially and the $f(\cdot)$ unit is reused d_m times to obtain the intermediate \boxplus sum S_m . Then, the outgoing messages R_{mn} ($\forall n \in \mathcal{N}_m$) are generated in a sequential order by the $g(\cdot)$ unit. Though the decoding is sequential for each check row, multiple (Z) check rows within one layer can be processed in parallel by employing multiple (Z) SISO decoders, which

* d_m is the number of non-zero elements in check row m .

increases the throughput by a factor of Z (see Fig. 5.3). Furthermore, the decoding throughput can be improved by overlapping the decoding of two layers as shown in Fig. 5.9. This scheduling would require dual-port memory for simultaneous read and write operations. Typically data dependencies between layers will occasionally stall the pipeline for one or more cycles. However the pipeline stalls can be avoided by shuffling the order of the layers [68].

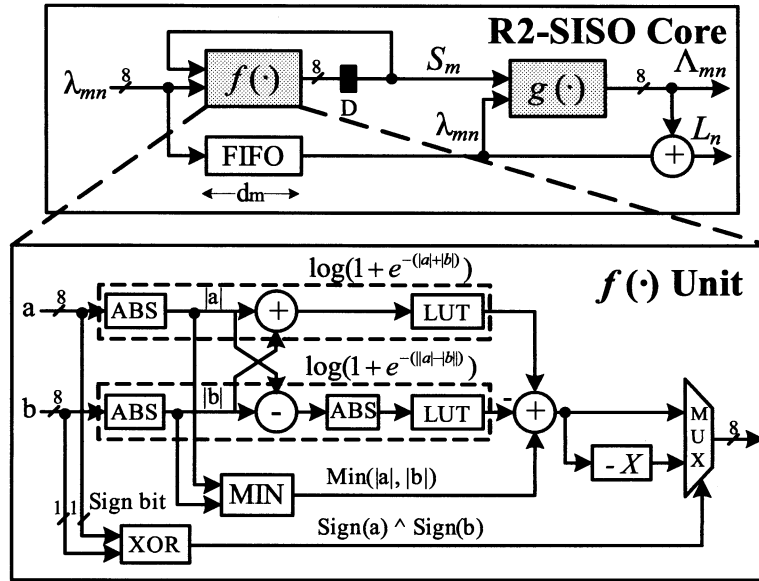


Figure 5.8 : Radix-2 (R2) SISO decoder architecture

5.5.3 Radix-4 SISO Decoder via Look-Ahead Transform

To increase the throughput of the R2-SISO decoder, a look-ahead transform can be used for the $f(\cdot)$ recursion. This transform leads to an increase in the number of data processed in each cycle as shown in Fig. 5.10, where two elements are processed

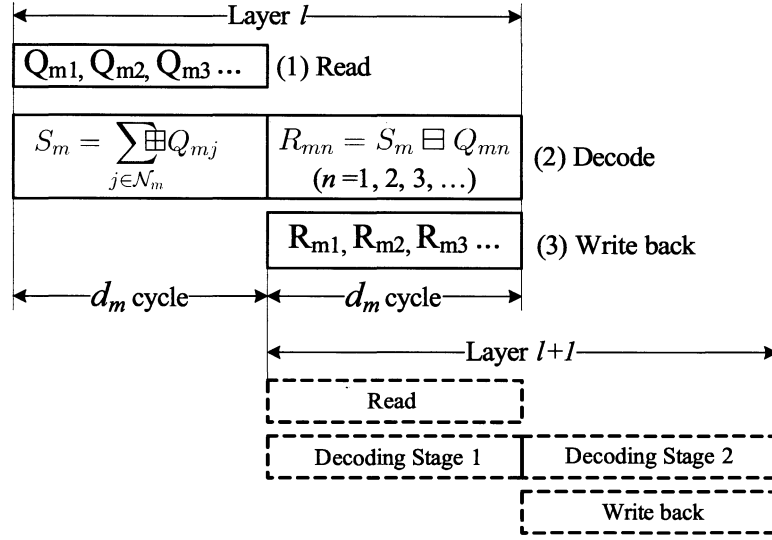
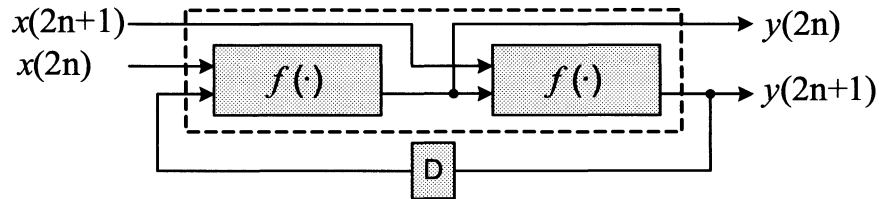


Figure 5.9 : Pipelined decoding schedule

in one clock cycle. We refer to this transform as Radix-4 (R4) recursion. Fig. 5.11 shows the corresponding Radix-4 SISO decoder architecture. Since two elements can be processed in each cycle, it has a throughput speed up of 2. Table 2 summarizes the synthesis results (90nm CMOS technology) for the R4 and R2 SISO decoders. To compare these two architectures, we define an efficiency factor η as the throughput speed-up with R4-SISO divided by the area overhead. As can be seen, R4-SISO achieves throughput-area efficiency gains especially at lower clock frequency.

Figure 5.10 : One level look-ahead transform of $f(\cdot)$ recursion

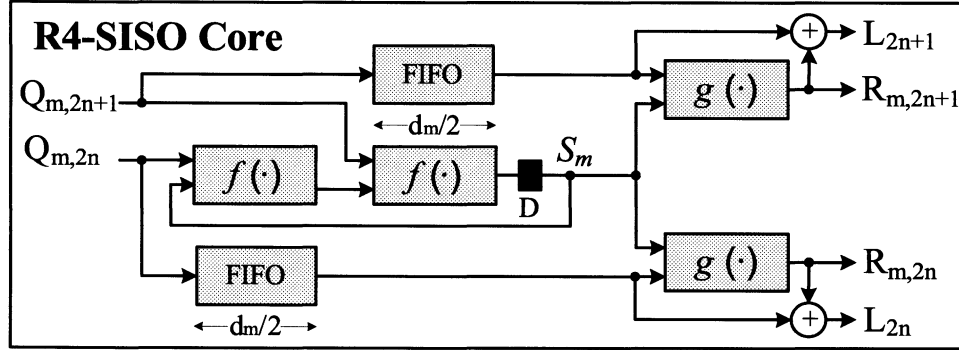


Figure 5.11 : Radix-4 (R4) SISO architecture

Table 2: Comparison of two SISO decoder architectures

	450 MHz	325 MHz	200 MHz
R2 SISO area	6978 μm^2	6367 μm^2	6197 μm^2
R4 SISO area	12774 μm^2	10077 μm^2	8944 μm^2
$\eta = \frac{\text{Speedup}}{\text{Area overhead}}$	1.09	1.26	1.39

5.5.4 Top Level Log-MAP LDPC Decoder Architecture

Fig. 5.12 shows the Log-MAP LDPC decoder architecture. In the proposed BS scheduling algorithm, the parallelism factor is equal to the sub-matrix size Z . Since parameter Z varies from code to code, i.e. 19 different sizes of Z are defined in WiMax, we must design a datapath that is modular and scalable to support different code types. This is achieved by employing distributed SISO decoders and memory banks as shown in Fig. 5.12. This architecture can also reduce the overall power consumption by deactivating the memory banks and SISO decoders that are not be-

ing used. The L messages, on the other hand, are stored in a central memory bank for parallel accessing by Z SISO decoders. This is achieved by grouping $[1 \times Z]$ L messages (associated with each sub-matrix) into one memory word.

The decoding flow for one sub-iteration is as follows: at each cycle, $[1 \times Z]$ L messages are first fetched from the L -memory and passed through a circular shifter to be routed to z SISO decoders. The soft input information Q_{mn} is formed by subtracting the old extrinsic message R_{mn} from the APP message L_n . Then the SISO decoder generates a new extrinsic message R_{mn} and APP message L'_n , and stores them back to the R -memory and the L -memory, respectively.

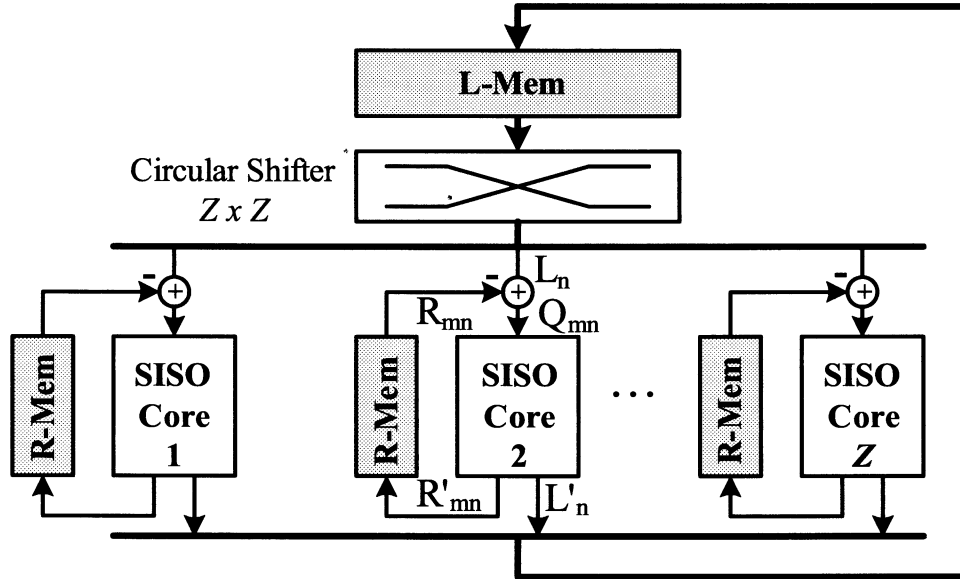


Figure 5.12 : Log-MAP LDPC decoder architecture with scalable datapath

By designing proper control logic, the decoder can be dynamically reconfigured to support multiple block-structured LDPC codes. With this partial-parallel architec-

ture, the pipelined (Radix-4) decoding throughput is approximately equal to:

$$\frac{2 \times N \times Z \times Rate \times fclk}{E \times I}, \quad (5.9)$$

where N is the number of block-columns in H , Z is the sub-matrix size, R is the code rate, E is the total number of non-zero sub-matrices in the parity check matrix, and I is the number of full iterations.

5.5.5 Performance Evaluation

The number of entries in the look-up-table determines the decoding performance and was analyzed in Fig. 5.13. We use two cases of IEEE 802.11n LDPC codes for simulation, and assume BPSK modulation and an AWGN channel with a (7.3) quantization scheme (7 total bits with 3 fractional bits). From Fig. 5.13, we can see that a 32-entry LUT has nearly no performance loss compared with the floating point belief propagation (BP). And a 24-entry LUT only has about 0.02dB performance loss compared with floating point BP. However a 16-entry LUT suffers about 0.05dB performance degradation. As a comparison, we also depict the performance of the offset min-sum approximation algorithm [63] which suffers 0.3 to 0.7dB performance degradation compared to floating point BP.

5.6 Multi-Layer Parallel LDPC Decoder Architecture

The conventional layered decoder architecture [71, 109] is initially developed to process the parity check matrix layer by layer, where each layer corresponds to a block-

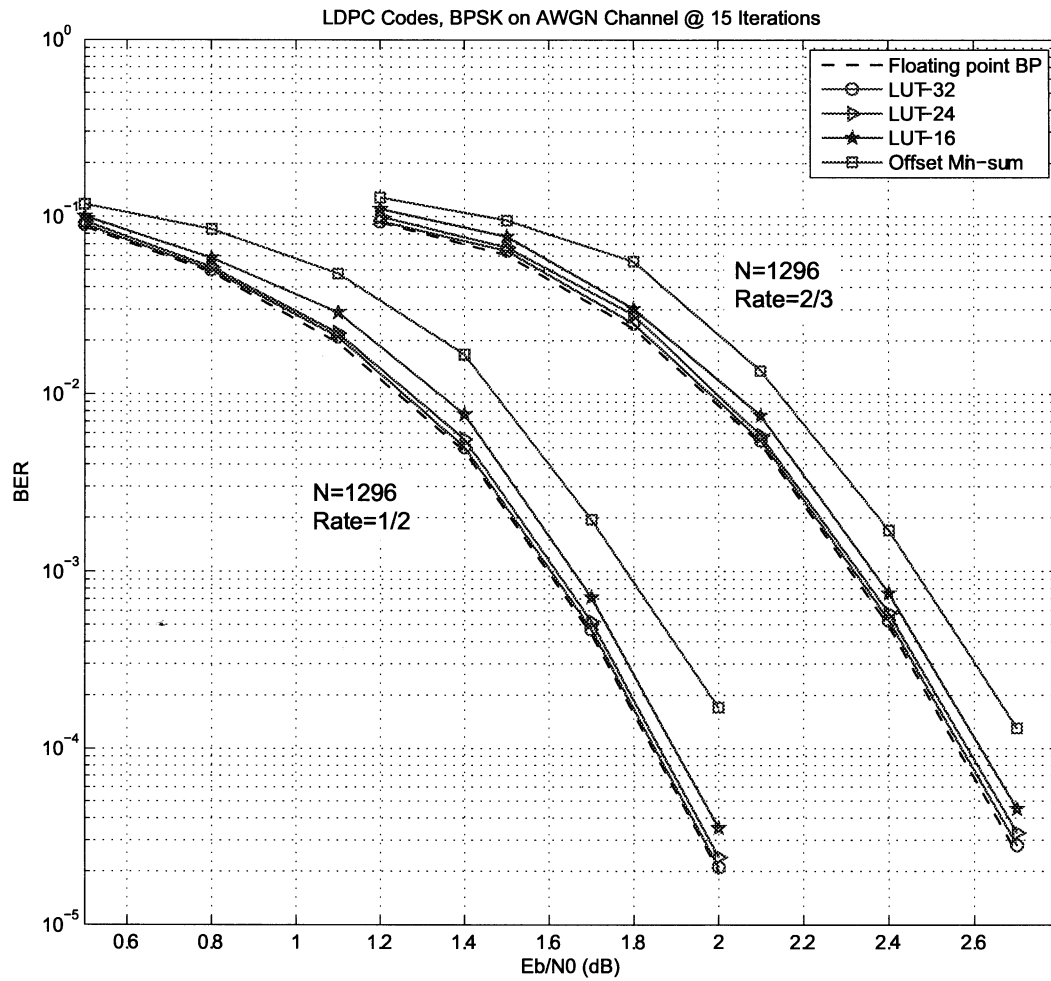


Figure 5.13 : Performance comparison of different LUT configurations.

row of the parity check matrix. Since the column-weight of each layer is typically 1 in many applications, such as IEEE 802.11n and IEEE 802.16e, this greatly simplifies the decoder design. To further improve the throughput, the two consecutive layers' data processing can be partially overlapped through a pipelined schedule [17, 65], where the data conflicts between two layers can be resolved by stalling the pipeline. The maximum row parallelism for the conventional layered algorithm is equal to the sub-matrix size Z , i.e. we can employ Z parallel check node processors to process Z rows in parallel. With this amount of parallelism, the conventional layered decoder can typically offer 100-1000 Mbps throughput [65, 68, 17, 122, 123].

To go beyond 1-Gbps throughput, the layered architecture needs to be extended to provide higher parallelism. One natural extension of the conventional layered architecture is to design a multi-layer parallel architecture where multiple (K) layers of a parity check matrix are processed in parallel. Now the maximum row parallelism is increased to KZ , i.e. we can employ KZ check node processors to process KZ rows in parallel. It should be noted that the multi-layer parallel decoding algorithm would still require less memory than the two-phase flooding algorithm because there is still no need to store the variable node messages in the multi-layered algorithm.

In this section, we propose a new multi-layer parallel decoding algorithm and VLSI architecture for high throughput LDPC decoding. The data conflicts between layers are resolved by modifying the LLR update rules. As a case study, we describe a double-layer parallel decoder architecture for IEEE 802.11n LDPC codes.

To support layer-level parallelism, we propose a multi-layer (K -layer) parallel decoding algorithm, where the maximum row parallelism is increased to KZ . When using the conventional layered algorithm to process multiple layers at the same time, data conflicts may occur when updating the LLRs because there can be more than one check node connected to a variable node. Fig. 5.14 shows an example of the data conflicts when updating LLRs for two consecutive layers, where check node (or row) m_0 and check node m_1 are both connected to variable node (or column) n . To resolve the data conflicts, we use the following LLR update rule for a K -layer parallel decoding algorithm. For a variable node n , let m_k represents the k -th check node that is connected to variable node n . Then the LLR value for variable node n is updated as:

$$L'_n = L_n + \sum_{k=0}^{K-1} (R'_{m_k,n} - R_{m_k,n}). \quad (5.10)$$

Compared to the original LLR update rule (5.6), the new LLR update rule combines all the check node messages and adds them to the old LLR value. We can define a macro-layer as a group of K layers of the parity check matrix. The multi-layer parallel decoding algorithm is summarized as follows. For each layer k in each macro-layer l , do the following:

$$Q_{m_k,n} = L_n - R_{m_k,n} \quad (5.11)$$

$$R'_{m_k,n} = S \cdot \prod_{j \in \mathcal{N}_{m_k} \setminus n} \text{sign}(Q_{m_k,j}) \cdot \min_{j \in \mathcal{N}_{m_k} \setminus n} |Q_{m_k,j}| \quad (5.12)$$

$$L'_n = L_n + \sum_{k=0}^{K-1} (R'_{m_k,n} - R_{m_k,n}). \quad (5.13)$$

In the above calculation, the LLR values L_n are updated macro-layer after macro-layer. Within each macro-layer, all the check rows can be processed in parallel, which therefore leads to a K times larger parallelism than the conventional layered algorithm. For example, we can use KZ number of check node processors to process KZ rows in parallel.

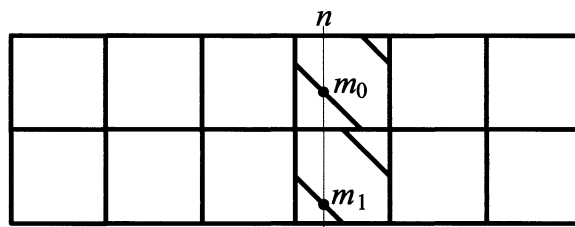


Figure 5.14 : Example of the data conflicts when updating LLRs for two layers.

5.6.1 Multi-Layer Decoding Performance Evaluation

In the multi-layer parallel decoding algorithm, the layer-parallelism K will have some negative impact on the decoding convergence speed because the LLR updates occur less frequently than in the single-layer algorithm. To compare the performance of the multi-layer parallel decoding algorithm against the conventional layered decoding algorithm, we perform floating-point simulations for the block length 1944 bits, code rate 1/2 IEEE 802.11n LDPC code. BPSK modulation is used for an AWGN channel. In the simulation, we collect at least 100 frame errors and the maximum iteration number is set to 15 for all the experiments. Fig. 5.15 compares the frame error rate (FER) performance of K -layer parallel decoders for $K = 1, 2, 3, 4, 6$. We also plot

the FER curve for the traditional two-phase flooding algorithm for comparison. As can be seen from the figure, the double-layer parallel decoder has shown a negligible performance loss, and the triple-layer parallel decoder has shown a small performance loss (< 0.1 dB). Compared with single layered decoding, as K increases, the FER performance slowly degrades as expected. Note that the performance loss can be compensated by slightly increasing the iteration number. Nevertheless, the K -layer parallel decoder will have a K -fold throughput increase compared to the conventional single-layer decoder. Note that compared to the two-phase flooding decoding, the throughput of the single-layered decoder is N times slower, where N is the total number of the layers. Thus, a trade-off can be made between the layer-parallelism K , the error performance, and the throughput.

5.6.2 Double-Layer Parallel Decoder Architecture for IEEE 802.11n LDPC Codes

As a case study, we have designed a double-layer parallel decoder for IEEE 802.11n LDPC codes. We propose a macroblock-serial (MB-serial) decoding algorithm. In this algorithm, a $Z \times Z$ sub-matrix is considered as a block and a macroblock (MB) contains four such blocks. Fig. 5.16(a) shows an example of an MB which contains four blocks: A, B, C, and D. Fig. 5.16(b) shows the MB view of the first two layers of the parity check matrix in Fig. 5.2. Because the rate $1/2$ matrix is sparser than the high rate matrix, some blocks in an MB can be zero blocks. However, for a denser

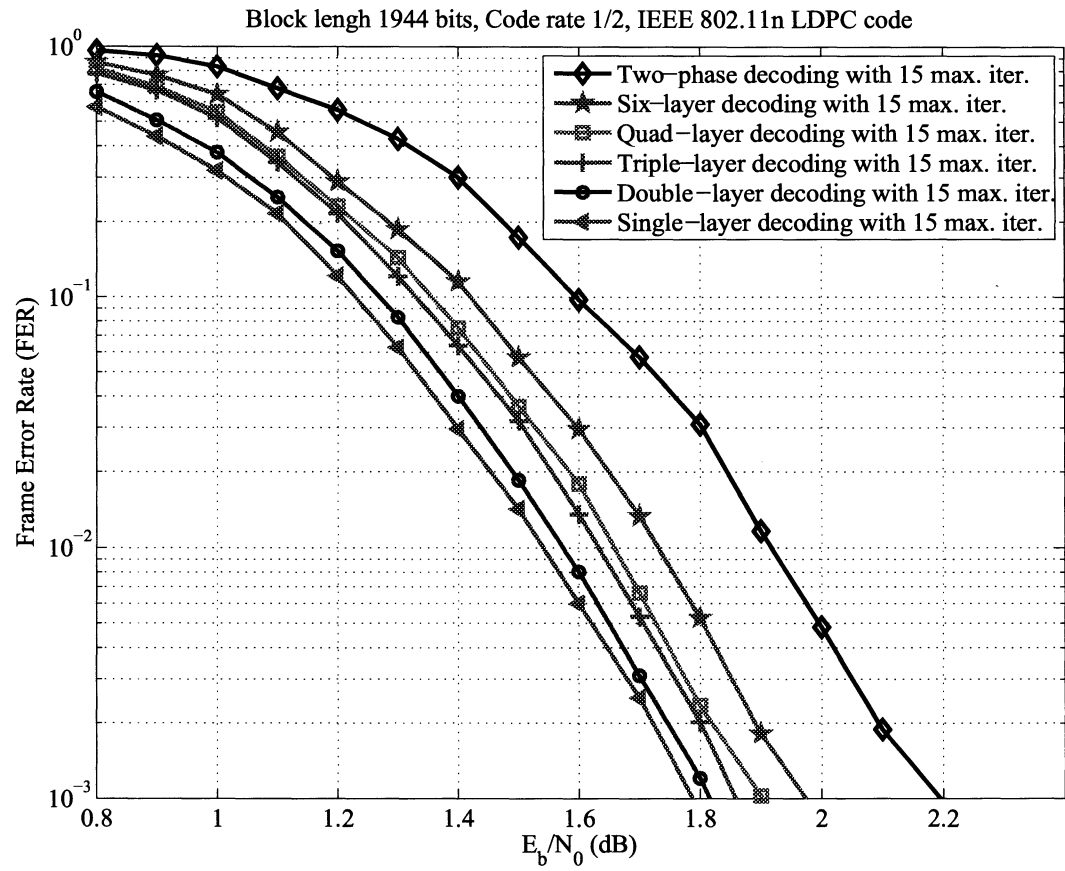


Figure 5.15 : Simulation results for multi-layer parallel decoding algorithm.

matrix, e.g. rate 5/6 matrix, all the four blocks in an MB are often non-zero blocks as shown in Fig. 5.16(c).

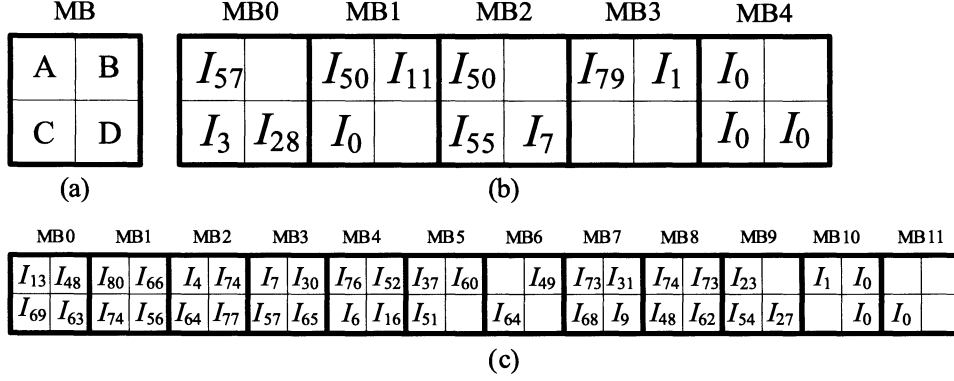


Figure 5.16 : (a) One MB with a dimension of $2Z \times 2Z$. (b) The MB view of the first two layers of the rate 1/2 matrix in Fig. 5.2. (c) The MB view of the first two layers of the matrix for rate 5/6, block length 1944 bits, 802.11n code.

We propose a partial parallel decoder architecture, where each MB is processed as a unit. Inside each macro-layer, MB is processed in serial, from left to right. Thus, we refer to this architecture as an MB-serial architecture. Fig. 5.17 shows the top level block diagram for the proposed MB-serial decoder architecture. In this architecture, the LLR memory is used for storing the initial and updated LLR values for each bit in a codeword. For LDPC codes with $M \times N$ sub-matrices each of which being a $Z \times Z$ shifted identity matrix, the LLR memory is organized such that Z LLR values are stored in the same memory word and there are N words in the memory. The LLR memory has two read-ports and two write-ports so that $2Z$ LLR values can be accessed at the same clock cycle. The decoding is a two-stage procedure. During the first stage, $2Z$ LLR values are read from the LLR memory at each clock cycle and

are passed to four permeters A, B, C, and D, which correspond to four blocks in an MB (cf. 5.16(a)). Note that for zero blocks in an MB, the corresponding permeters and other related logic will be disabled.

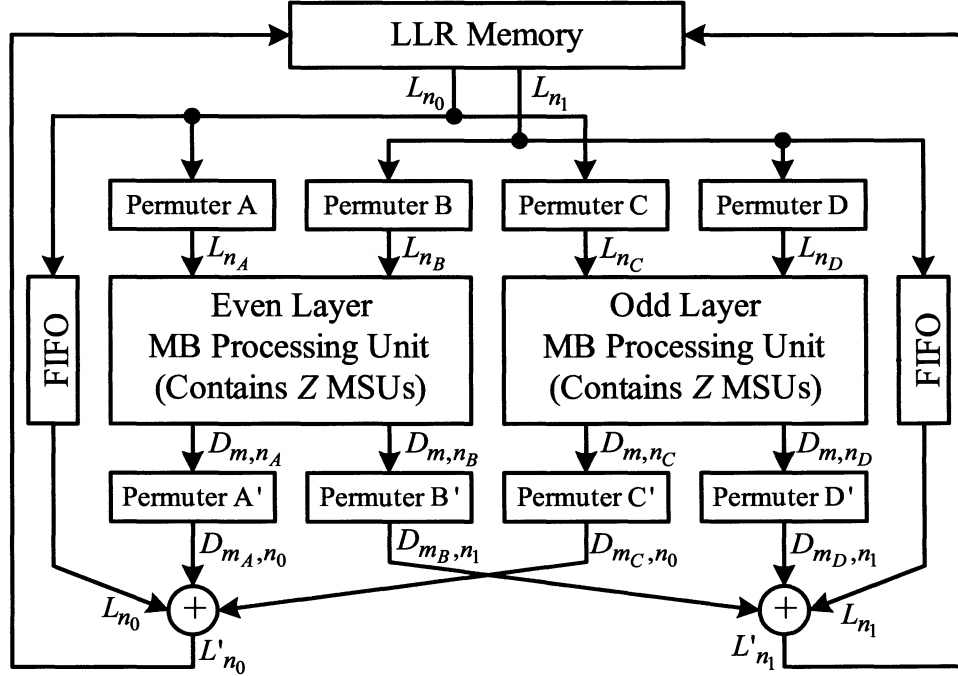


Figure 5.17 : MB-serial LDPC decoder architecture for the double-layer example.

The $2Z$ permuted LLR values L_{n_A} and L_{n_B} are fed to the even-layer's MB processing unit, and the other $2Z$ permuted LLR values L_{n_C} and L_{n_D} are fed to the odd-layer's MB processing unit. Each MB processing unit consists of $Z = 81$ min-sum units (MSUs) based on the maximum sub-matrix size defined in the IEEE 802.11n standard. Fig. 5.18 shows the block diagram for one MSU. Each MSU can process two LLR values at each clock cycle so that altogether Z MSUs can process $2Z$ LLR values at each clock cycle. During the first stage, Q values are computed by subtract-

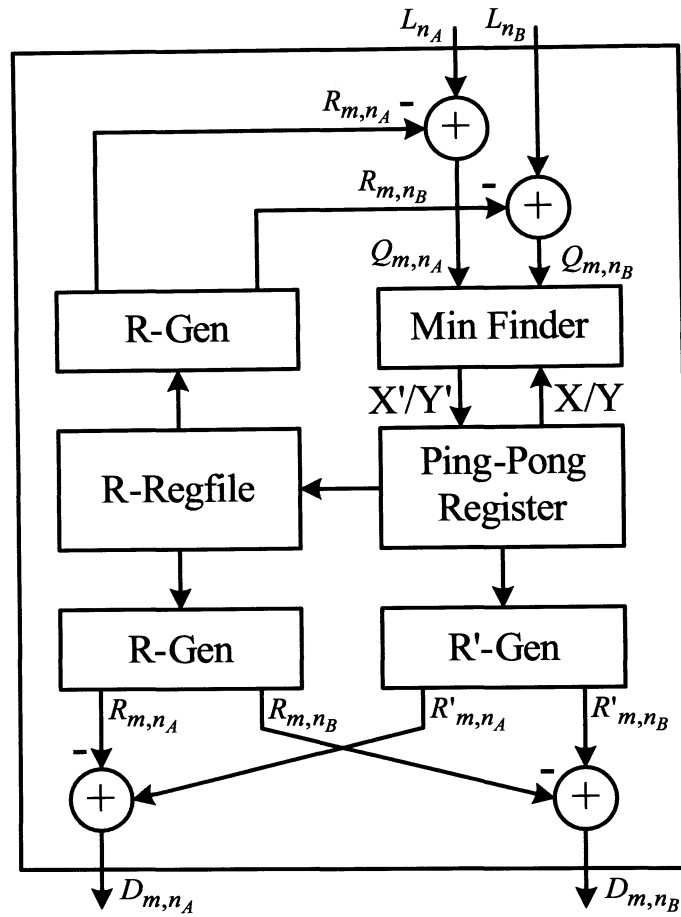


Figure 5.18 : Block diagram for the pipelined Min-sum unit (MSU).

Index = Super-layer number

0	Min 0	Min 1	Pos	Sign Array
1	Min 0	Min 1	Pos	Sign Array
\vdots
$M/2-1$	Min 0	Min 1	Pos	Sign Array

Figure 5.19 : R-Regfile organization.

ing the R values from the LLR values based on (5.11). The R values are stored in a compressed way. The R-Regfile is used to store the information for restoring the $R_{m,n}$ values. Fig. 5.19 shows the organization of the R-Regfile. For each row m , only the first minimum (min0), the second minimum (min1), the position of the first minimum (pos), and the sign bits for all Q_{m,n_j} related to row m are stored in the R-Regfile. A R value generator (R-Gen) is used to restore the R values from the R-Regfile as:

$$|R_{m,n_j}| = \begin{cases} 0.75Y_m, & \text{if } n_j = P_m \\ 0.75X_m, & \text{otherwise,} \end{cases} \quad (5.14)$$

where X_m and Y_m denote the first minimum value and the second minimum value for row m , respectively, and P_m denotes the position of the first minimum value for row m . The sign bits of the R_{m,n_j} value are generated using the sign array. As the scaled min-sum algorithm is used, the R value is scaled by a factor of 0.75. A min finder unit (MFU) is used to compare the Q_{m,n_A} and Q_{m,n_B} values against X and Y read from the Ping-Pong register, where X and Y are the first minimum and the second minimum temporary variables and are initialized to be the maximum possible positive values. The two new minimum values X' and Y' are stored in the Ping-Pong register. The index of the minimum Q value and sign bits for all Q values are also updated in the Ping-Pong register. The Ping-Pong register consists of two registers (ping and pong registers), where each register has the same organization as one word of the R-Regfile. Two registers are required because we want to support pipelined decoding by overlapping two macro-layers' data processing. During the second stage,

the R'-Gen unit gets values from the Ping-Pong register and restores the most recently updated R' values. Another R-Gen unit gets values from R-Regfile and restores the old R values. Then a Delta-R value, denoted as D value, is formed by:

$$D_{m,n_j} = R'_{m,n_j} - R_{m,n_j}. \quad (5.15)$$

The R-Regfile has two read-ports so that it can be accessed simultaneously by two consecutive macro-layers. After the second stage, the contents of the Ping-Pong register is written to the R-Regfile overwriting the values for the current macro-layer, and the Ping and Pong registers switch role.

Now turning back to the top level decoder in Fig. 5.17, after the $2Z$ D values are produced by each MB processing unit, the D values are de-permuted and added to the LLR values from the FIFO to form the updated $2Z$ LLR values as:

$$L'_{n_0} = L_{n_0} + D_{m_A,n_0} + D_{m_C,n_0} \quad (5.16)$$

$$L'_{n_1} = L_{n_1} + D_{m_B,n_1} + D_{m_D,n_1}. \quad (5.17)$$

The new updated LLR values are then written back to the LLR memory.

To further increase the throughput, we can overlap the decoding process of two macro-layers. The pipelined data flow is illustrated in Fig. 5.20. The data dependencies between two macro-layers are avoided by using a scoreboard to keep track of the read and write sequences of the LLR values. Pipeline stalls will be inserted if there is a data dependency between two macro-layers. If one ignores the extra pipeline stalls, which are typically small, the proposed double-layer pipelined decoder can process

two macro-layers of the matrix simultaneously, which leads to a significant throughput improvement.

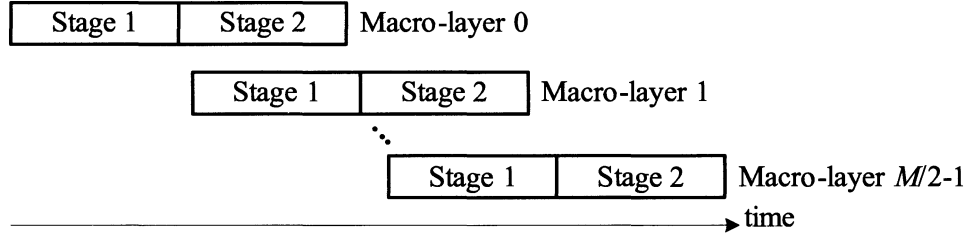


Figure 5.20 : Pipelined decoding data flow for the double-layer example.

It should be noted that the described double-layer parallel architecture shown in Fig. 5.17 can be generalized for a K -layer parallel architecture by employing K macroblock processing units to process K layers in parallel.

5.7 Discussion on the Similarities of LDPC Decoders and Turbo Decoders

LDPC codes and Turbo codes have many similarities, e.g. they all have a trellis structure that can be processed using a similar MAP algorithm [14]. We can develop a specialized decoder for each family for higher performance. We can also develop a configurable decoder for both families of codes with limited hardware overhead. For example, we can extend the single-layered LDPC decoder architecture to support Turbo codes. Recall that in Chapter 4, we have presented a parallel Turbo decoder based on multiple MAP units. We can develop a unified MAP unit for both LDPC

codes and Turbo codes.

5.8 Flexible and Configurable LDPC/Turbo Decoder

In this section, we propose a unified decoding algorithm for both LDPC codes and Turbo codes. We extend the layered LDPC decoder architecture to support Turbo codes with a low hardware overhead.

5.8.1 Flex-SISO Module

To support both LDPC codes and Turbo codes, usually two separate decoders are needed. To save area, we propose a flexible soft-input soft-output (SISO) module, named Flex-SISO module, for decoding of both LDPC and Turbo codes. The SISO module is based on the MAP algorithm [91]. To reduce complexity, the MAP algorithm is usually calculated in the log domain [89]. In this thesis, we assume the MAP algorithm is always calculated in the log domain.

The decoding algorithm underlying the Flex-SISO module works for codes which have trellis representations. For LDPC codes, a Flex-SISO module was used to decode a layer of a parity check matrix, or super-code. For Turbo codes, a Flex-SISO module was used to decode a component convolutional code. The iteration performed by the Flex-SISO module is called a sub-iteration, and thus one full iteration contains n sub-iterations.

Fig. 5.21 depicts the proposed Flex-SISO module. The output of the Flex-SISO

module is the *a posteriori* probability (APP) log-likelihood ratio (LLR) values, denoted as $\lambda_o(u)$, for information bits. It should be noted that the Flex-SISO module exchanges the soft values $\lambda_o(u)$ instead of the extrinsic values in the iterative decoding process. The extrinsic values, denoted as $\lambda_e(u)$, are stored in a local memory of the Flex-SISO module. To distinguish the extrinsic values generated at different sub-iterations, we use $\lambda_e(u; old)$ and $\lambda_e(u; new)$ to represent the extrinsic values generated in the previous sub-iteration and the current sub-iteration, respectively. The soft input values $\lambda_i(u)$ are the outputs from the previous Flex-SISO module, or other previous modules if necessary. Another input to the Flex-SISO module is the channel values for parity bits, denoted as $\lambda_c(p)$, if available. For LDPC codes, we do not distinguish information and parity bits, and all the codeword bits are treated as information bits. However, in the case of Turbo codes, we treat information and parity bits separately. Thus the input port $\lambda_c(p)$ will not be used when decoding of LDPC codes. At each sub-iteration, the old extrinsic values, denoted as $\lambda_e(u; old)$, are retrieved from the local memory and should be subtracted from the soft input values $\lambda_i(u)$ to avoid positive feedback.

A generic description of the message passing algorithm is as follows. Multiple Flex-SISO modules are connected in series to form an iterative decoder. First, the Flex-SISO module receives the soft values $\lambda_i(u)$ from upstream Flex-SISO modules and the channel values (for parity bits) $\lambda_c(p)$ if available. The $\lambda_i(u)$ can be thought of as the sum of the channel value $\lambda_c(u)$ (for information bit) and all the extrinsic

values $\lambda_e(u)$ previously generated by all the super-codes:

$$\lambda_i(u) = \lambda_c(u) + \sum \lambda_e(u). \quad (5.18)$$

Note that prior to the iterative decoding, $\lambda_i(u)$ should be initialized with $\lambda_c(u)$. Next, the old extrinsic value $\lambda_e(u; old)$ generated by this Flex-SISO module in the previous iteration is subtracted from $\lambda_i(u)$ as follows:

$$\lambda_t(u) = \lambda_i(u) - \lambda_e(u; old). \quad (5.19)$$

Then, the new extrinsic value $\lambda_e(u; new)$ can be computed using the MAP algorithm based on $\lambda_t(u)$, and $\lambda_c(p)$ if available. Finally, the APP value is updated as

$$\lambda_o(u) = \lambda_i(u) - \lambda_e(u; old) + \lambda_e(u; new). \quad (5.20)$$

Then this updated APP value is passed to the downstream Flex-SISO modules. This computation repeats in each sub-iteration.

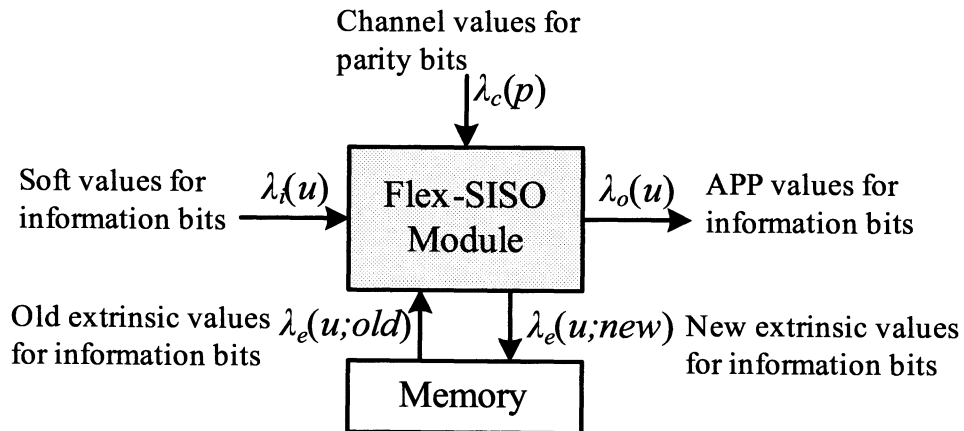


Figure 5.21 : Flex-SISO module.

5.8.2 Flex-SISO Module to Decode LDPC Codes

In this section, we show how to use the Flex-SISO module to decode LDPC codes. Because QC-LDPC codes are widely used in many practical systems, we will primarily focus on the QC-LDPC codes. First, we decompose a QC-LDPC code into multiple super-codes, where each layer of the parity check matrix defines a super-code. After the layered decomposition, each super-code comprises z independent 2-state single parity check codes. Fig. 5.22 shows the super-code based, or layered, LDPC decoder architecture based on the Flex-SISO modules. The decoder parallelism at each Flex-SISO module is at the level of the sub-matrix size z , because these z single parity codes have no data dependency and can thus be processed simultaneously. This architecture differs from the regular two-phase flooding LDPC decoder in that a code is partitioned into multiple sections, and each section is processed by the same processor. This scheduling algorithm is similar to the layered scheduling algorithm [71]. The convergence rate can be twice faster than that of a regular decoder.

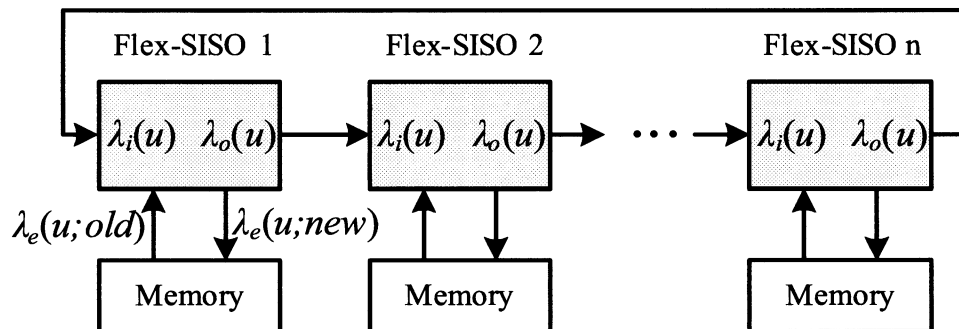


Figure 5.22 : LDPC decoding using Flex-SISO modules where a LDPC code is decomposed into n super-codes, and n Flex-SISO modules are connected in series to decode.

Since the data flow is the same between different sub-iterations, one physical Flex-SISO module is instantiated, and it is re-used at each sub-iteration, which leads to a partial-parallel decoder architecture. Fig. 5.23 shows an iterative LDPC decoder hardware architecture based on the Flex-SISO module. The structure comprises an APP memory to store the soft APP values, an extrinsic memory to store the extrinsic values, and a MAP processor to implement the MAP algorithm for z single parity check codes. Prior to the iterative decoding process, the APP memory is initialized with channel values $\lambda_c(u)$, and the extrinsic memory is initialized with 0.

The decoding flow is summarized as follows. It should be noted that the parity bits are treated as information bits for the decoding of LDPC codes. We use the symbol u_k to represent the k -th data bit in the codeword. For check node m , we use the symbol $u_{m,k}$ to denote the k -th codeword bit (or variable node) that is connected to this check node m . To remove correlations between iterations, the old extrinsic message is subtracted from the soft input message to create a temporary message λ_t as follows

$$\lambda_t(u_{m,k}) = \lambda_i(u_k) - \lambda_e(u_{m,k}; old), \quad (5.21)$$

where $\lambda_i(u_k)$ is the soft input log likelihood ratio (LLR) and $\lambda_e(u_{m,k}; old)$ is the old extrinsic value generated by this MAP processor in the previous iteration. Then the new extrinsic value can be computed as:

$$\lambda_e(u_{m,k}; new) = \sum_{j:j \neq k} \boxplus \lambda_t(u_{m,j}), \quad (5.22)$$

where the \boxplus operation is associative and commutative, and is defined as [120]

$$\lambda(u_1) \boxplus \lambda(u_2) = \log \frac{1 + e^{\lambda(u_1)} e^{\lambda(u_2)}}{e^{\lambda(u_1)} + e^{\lambda(u_2)}}. \quad (5.23)$$

Finally, the new APP value is updated as:

$$\lambda_o(u_k) = \lambda_t(u_{m,k}) + \lambda_e(u_{m,k}; new). \quad (5.24)$$

For each sub-iteration l , equations (5.21-5.24) can be executed in parallel for check nodes $m = lz$ to $lz + z - 1$ because there are no data dependency between them.

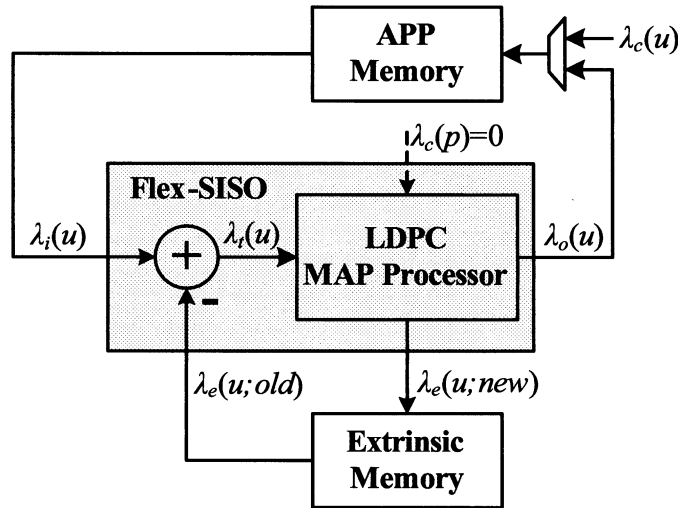


Figure 5.23 : LDPC decoder architecture based on the Flex-SISO module.

5.8.3 Flex-SISO Module to Decode Turbo Codes

In this section, we show how to use the Flex-SISO module to decode Turbo codes. A Turbo code can be naturally partitioned into two super-codes, or constituent codes. In a traditional Turbo decoder, where the extrinsic messages are exchanged between

two super-codes, the Flex-SISO module can not be directly applied, because the Flex-SISO module requires the APP values, rather than the extrinsic values, being exchanged between super-codes. In this section, we made a small modification to the traditional Turbo decoding flow so that the APP values are exchanged in the decoding procedure.

The traditional Turbo decoding procedure with two SISO decoders is shown in Fig. 5.24. The definitions of the symbols in the figure are as follows. The information bit and the parity bits at time k are denoted as u_k and $(p_k^{(1)}, p_k^{(2)}, \dots, p_k^{(n)})$, respectively, with $u_k, p_k^{(i)} \in \{0, 1\}$. The channel LLR values for u_k and $p_k^{(i)}$ are denoted as $\lambda_c(u_k)$ and $\lambda_c(p_k^{(i)})$, respectively. The *a priori* LLR, the extrinsic LLR, and the APP LLR for u_k are denoted as $\lambda_a(u_k)$, $\lambda_e(u_k)$, and $\lambda_o(u_k)$, respectively.

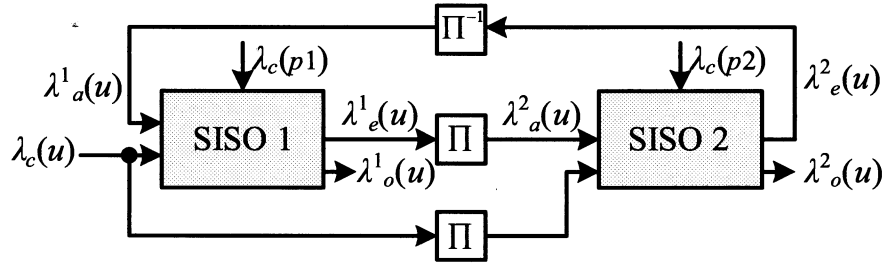


Figure 5.24 : Traditional Turbo decoding procedure using two SISO decoders, where the *extrinsic* LLR values are exchanged between two SISO decoders.

In the decoding process, the SISO decoder computes the extrinsic LLR value at time k as follows:

$$\begin{aligned} \lambda_e(u_k) = & \max_{\mathbf{u}: u_k=1}^* \{ \alpha_{k-1}(s_{k-1}) + \gamma_k^e(s_{k-1}, s_k) + \beta_k(s_k) \} \\ & - \max_{\mathbf{u}: u_k=0}^* \{ \alpha_{k-1}(s_{k-1}) + \gamma_k^e(s_{k-1}, s_k) + \beta_k(s_k) \}. \end{aligned} \quad (5.25)$$

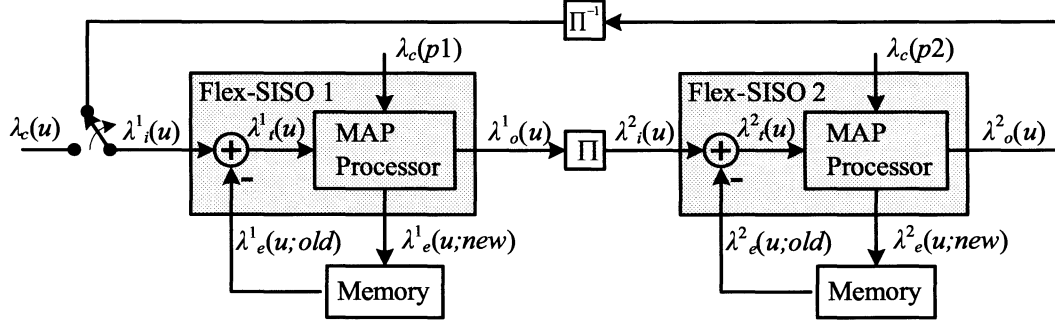


Figure 5.25 : Modified Turbo decoding procedure using two Flex-SISO modules. The *soft* LLR values are exchanged between two SISO modules.

The α and β metrics are computed based on the forward and backward recursions:

$$\alpha_k(s_k) = \max_{s_{k-1}}^* \{ \alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) \} \quad (5.26)$$

$$\beta_k(s_k) = \max_{s_{k+1}}^* \{ \beta_{k+1}(s_{k+1}) + \gamma_k(s_k, s_{k+1}) \}, \quad (5.27)$$

where the branch metric γ_k is computed as:

$$\gamma_k = u_k \cdot (\lambda_c(u_k) + \lambda_a(u_k)) + \sum_i^n p_k^{(i)} \cdot \lambda_c(p_k^{(i)}). \quad (5.28)$$

The *extrinsic* branch metric γ_k^e in (5.25) is computed as:

$$\gamma_k^e = \sum_i^n p_k^{(i)} \cdot \lambda_c(p_k^{(i)}). \quad (5.29)$$

The $\max^*(\cdot)$ function in (5.25-5.27) is defined as:

$$\max^*(a, b) = \max(a, b) + \log(1 + e^{-|a-b|}). \quad (5.30)$$

The soft APP value for u_k is generated as:

$$\lambda_o(u_k) = \lambda_e(u_k) + \lambda_a(u_k) + \lambda_c(u_k). \quad (5.31)$$

In the first half iteration, SISO decoder 1 computes the extrinsic value $\lambda_e^1(u_k)$ and passes it to SISO decoder 2. Thus, the extrinsic value computed by SISO decoder 1 becomes the *a priori* value $\lambda_a^2(u_k)$ for SISO decoder 2 in the second half iteration. The computation is repeated in each iteration. The iterative process is usually terminated after a certain number of iterations, when the soft APP value $\lambda_o(u_k)$ converges.

Modified Turbo Decoder Structure Using Flex-SISO Modules

In order to use the proposed Flex-SISO module for Turbo decoding, we modify the traditional Turbo decoder structure. Fig. 5.25 shows the modified Turbo decoder structure based on the Flex-SISO modules.

It should be noted that the modified Turbo decoding flow is mathematically equivalent to the original Turbo decoding flow, but uses a different message passing method. The modified data flow is as follows. In the first half iteration, Flex-SISO decoder 1 receives soft LLR value $\lambda_i^1(u_k)$ from Flex-SISO decoder 2 through de-interleaving ($\lambda_i^1(u_k)$ is initialized to channel value $\lambda_c(u_k)$ prior to decoding). Then it removes the old extrinsic value $\lambda_e^1(u_k; old)$ from the soft input LLR $\lambda_i^1(u_k)$ to form a temporary message $\lambda_t^1(u_k)$ as follows (for brevity, we drop the superscript “1” in the following equations)

$$\lambda_t(u_k) = \lambda_i(u_k) - \lambda_e(u_k; old). \quad (5.32)$$

To relate to the traditional Turbo decoder structure, this temporary message is mathematically equal to the sum of the channel value $\lambda_c(u_k)$ and the *a priori* value $\lambda_a(u_k)$

in Fig. 5.24:

$$\lambda_t(u_k) = \lambda_c(u_k) + \lambda_a(u_k). \quad (5.33)$$

Thus, the branch metric calculation in (5.28) can be re-written as:

$$\gamma_k = u_k \cdot \lambda_t(u_k) + \sum_i^n p_k^{(i)} \cdot \lambda_c(p_k^{(i)}). \quad (5.34)$$

The extrinsic branch metric (γ_k^e) calculation, and the extrinsic LLR ($\lambda_e(u_k)$) calculation, however, remain the same as (5.29) and (5.25-5.27), respectively. Finally, the soft APP LLR output is computed as:

$$\lambda_o(u_k) = \lambda_t(u_k) + \lambda_e(u_k; new). \quad (5.35)$$

In the Flex-SISO based iterative decoding procedure, the soft outputs $\lambda_o^1(u)$ computed by Flex-SISO decoder 1 are passed to Flex-SISO decoder 2 so that they become the soft inputs $\lambda_i^2(u)$ for Flex-SISO decoder 2 in the second half iteration. The computation is repeated in each half-iteration until the iteration converges. Since the operations are identical between two sub-iterations, only one physical Flex-SISO module is instantiated, and it is re-used for two sub-iterations.

Fig. 5.26 shows an iterative Turbo decoder architecture based on the Flex-SISO module. The architecture is very similar to the LDPC decoder architecture shown in Fig. 5.23. The main differences are: 1) the Turbo decoder has separate parity channel LLR inputs whereas the LDPC decoder treats parity bits as information bits, 2) the Turbo decoder employs the MAP algorithm on an N -state trellis whereas the LDPC decoder applies the MAP algorithm on z independent 2-state trellises, and 3) the

interleaver/permuter structures are different (not shown in the figures). But despite these differences, there are certain important commonalities. The message passing flows are the same. The memory organizations are similar, but with a variety of sizes depending on the codeword length. The MAP processors, which will be described in the next section, have similar functional unit resources that will be configured using multiplexors for each algorithm. Thus, it is natural to design a unified SISO decoder with configurable MAP processors to support both LDPC and Turbo codes.

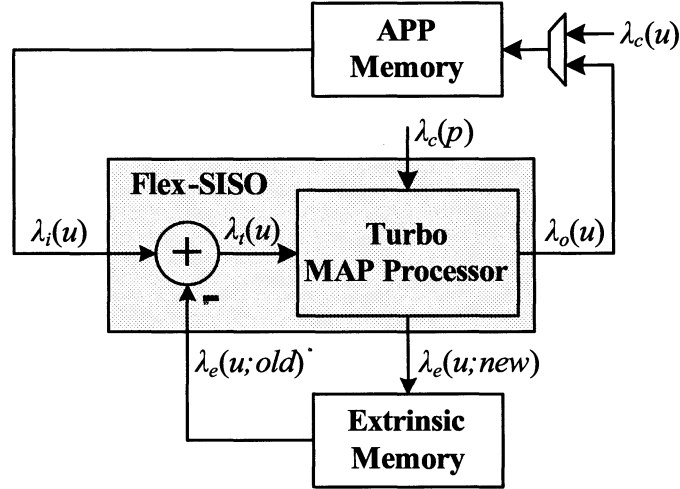


Figure 5.26 : Turbo decoder architecture based on the Flex-SISO module.

5.8.4 Design of A Flexible Functional Unit

The MAP processor is the main processing unit in both LDPC and Turbo decoders as depicted in Fig. 5.23 and Fig. 5.26. In this section, we introduce a flexible functional unit to decode LDPC and Turbo codes with a small additional overhead.

MAP Functional Unit for Turbo Codes

In a Turbo MAP processor, the critical path lies in the state metric calculation unit which is often referred to as add-compare-select-add (ACSA) unit. As depicted in Fig. 5.27, for each state m of the trellis, the decoder needs to perform an ACSA operation as follows:

$$\alpha'_0 = \max^*(\alpha_0 + \gamma_0, \alpha_1 + \gamma_1), \quad (5.36)$$

where α_0 and α_1 are the previous state metrics, and γ_0 and γ_1 are the branch metrics. Fig. 5.27(b) shows a circuit implementation for the ACSA unit, where a signed-input look-up table “LUT-S” was used to implement the non-linear function $\log(1 + e^{-|x|})$. This circuit can be used to recursively compute the forward and backward state metrics based on eq. (5.26)(5.27).

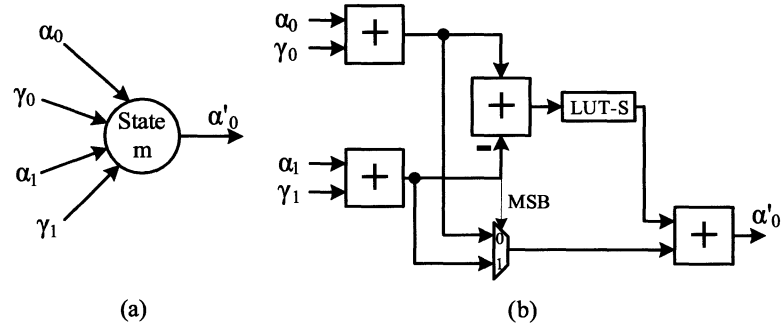


Figure 5.27 : Turbo ACSA structure. (a) Flow of state metric calculation. (b) Circuit diagram for the Turbo ACSA unit.

MAP Functional Unit for LDPC Codes

In the layered QC-LDPC decoding algorithm, each super-code comprises z independent single parity check codes. Each single parity check code can be viewed as a terminated 2-state convolutional code. Fig. 5.28 shows an example of the trellis structure for a single parity check node.

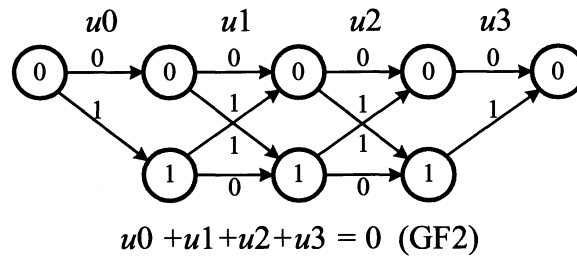


Figure 5.28 : Trellis structure for a single parity check code.

An efficient MAP decoding algorithm for a single parity check code was given in [124]: for independent random variables u_0, u_1, \dots, u_l the extrinsic LLR value for bit u_k is computed as:

$$\lambda(u_k) = \sum_{\sim\{u_k\}} \boxplus \lambda_i(u_i), \quad (5.37)$$

where the compact notation $\sim\{u_k\}$ represents the set of all the variables with u_k excluded. For brevity, we define a function $f(a, b)$ to represent the operation $\lambda_i(u_1) \boxplus \lambda_i(u_2)$ as follows

$$f(a, b) = \log \frac{1 + e^a e^b}{e^a + e^b}, \quad (5.38)$$

where $a \triangleq \lambda_i(u_1)$ and $b \triangleq \lambda_i(u_2)$. Fig. 5.29 shows a forward-backward decoding flow

to implement (5.37). The forward (α) and backward (β) recursions are defined as:

$$\alpha_{k+1} = f(\alpha_k, \gamma_k) \quad (5.39)$$

$$\beta_k = f(\beta_{k+1}, \gamma_{k+1}), \quad (5.40)$$

where $\gamma_k = \lambda_i(u_k)$ and is referred to as the branch metric as an analogy to a Turbo decoder. The α and β metrics are initialized to $+\infty$ in the beginning. Based on the α and β metrics, the extrinsic LLR for u_k is computed as:

$$\lambda(u_k) = f(\alpha_k, \beta_k). \quad (5.41)$$

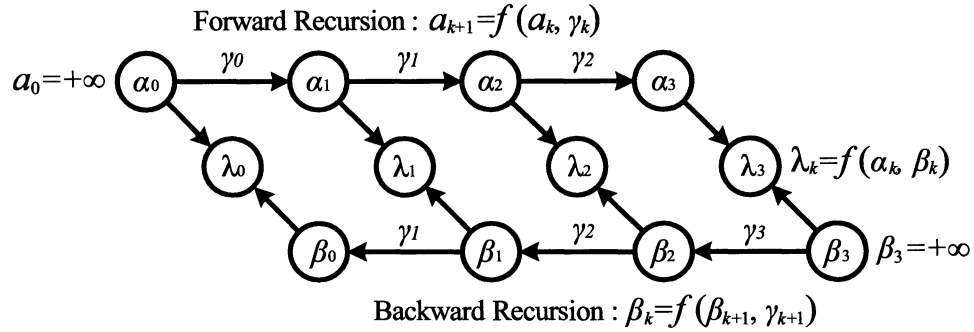


Figure 5.29 : A forward-backward decoding flow to compute the extrinsic LLRs for single parity check code.

Fig. 5.30 shows a MAP processor structure to decode the single parity check code. Three identical $f(a, b)$ units are used to compute α , β , and λ values. To relate to the top level LDPC decoder architecture as shown in Fig. 5.23, the inputs to this MAP processor are the temporary metrics $\lambda_t(u_{m,k})$, and the outputs from this MAP processor are the extrinsic metrics $\lambda_e(u_{m,k}; new)$.

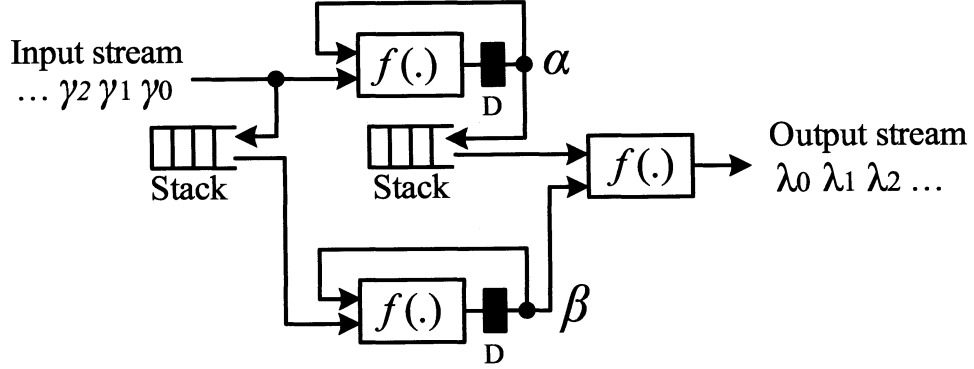


Figure 5.30 : MAP processor structure for single parity check code.

To compute (5.38) in hardware, we separate the operation into sign and magnitude calculations:

$$\begin{aligned} \text{sign}(f(a, b)) &= \text{sign}(a) \text{sign}(b), \\ |f(a, b)| &= \min(|a|, |b|) + \log(1 + e^{-(|a|+|b|)}) \\ &\quad - \log(1 + e^{-||a|-|b||}). \end{aligned} \quad (5.42)$$

Compared to the classical “tanh” function used in LDPC decoding

$$\Psi(x) = -\log(\tanh(|x/2|)), \quad (5.43)$$

the $f(\cdot)$ function is numerically more robust and less sensitive to quantization noise. Due to its widely dynamic range (up to $+\infty$), the $\Psi(x)$ function has a high complexity and is prone to quantization noise. Although many approximations have been proposed to improve the numerical accuracy of $\Psi(x)$ [125, 126, 72], it is still expensive to implement the $\Psi(x)$ function in hardware. However, the non-linear term in the

$f(\cdot)$ function has a very small dynamic range:

$$0 < g(x) \triangleq \log(1 + e^{-|x|}) < 0.7,$$

thus the $f(\cdot)$ function can be more easily implemented in hardware by using a low complexity look-up table (LUT). To implement $g(x)$ in hardware, we propose to use a 4-value LUT approximation which is shown in table 5.1. For fixed point implementation, we propose to use 2 fractional bits to implement the LUT. Table 5.2 shows the proposed LUT implementation. It should be noted that $g(x)$ is the same as the non-linear term in the Turbo $\max^*(\cdot)$ function (c.f. eq. (5.30)). Thus, the same look-up table configuration can be applied to the Turbo ACSA unit.

Table 5.1 : LUT approximation for $g(x) = \log(1 + e^{-|x|})$

$ x $	$ x = 0$	$0 < x \leq 0.75$	$0.75 < x \leq 2$	$ x > 2$
$g(x)$	0.75	0.5	0.25	0

Table 5.2 : LUT implementation

$ x $	0	1	2	3	4	5	6	7	8	> 8
$g(x)$	3	2	2	2	1	1	1	1	1	0

Fig. 5.31 depicts a circuit implementation for the LDPC $|f(a, b)|$ functional unit using two look-up tables “LUT-S” and “LUT-U”, where LUT-S and LUT-U imple-

ment $\log(1 + e^{-|a|-|b|})$ and $\log(1 + e^{-(|a|+|b|)})$, respectively. The difference between LUT-S and LUT-U is that: LUT-S is a signed-input look-up table that takes both positive and negative data inputs whereas LUT-U is an unsigned-input look-up table (half size of LUT-S) that only takes positive data inputs.

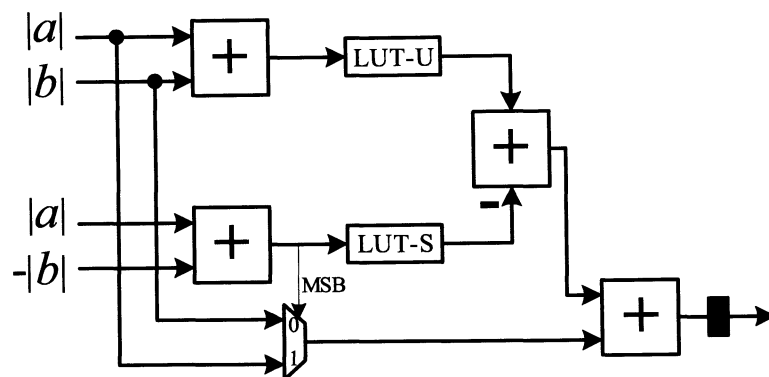


Figure 5.31 : Circuit diagram for the LDPC $|f(a, b)|$ functional unit.

Unified MAP Functional Unit

If we compare the LDPC $|f(a, b)|$ functional unit (c.f. Fig. 5.31) with the Turbo ACSA functional unit (c.f. Fig. 5.27), we can see that they have many commonalities except for the position of the look-up tables and the multiplexor. To support both LDPC and Turbo codes with minimum hardware overhead, we propose a flexible functional unit (FFU) which is depicted in Fig. 5.32. We modify the look-up table structure so that each look-up table can be bypassed when the *bypass* control signal is high. A *select* signal was used to switch between the LDPC mode and the Turbo

Table 5.3 : Functional description of the FFU

Signals	LDPC Mode	Turbo Mode
<i>select</i>	1	0
<i>bypass1</i>	0	1
<i>bypass2</i>	1	0
X	$ a $	α_0
Y	$ b $	γ_0
V	$ a $	α_1
W	$- b $	γ_1
Z	$ f(a, b) $	$\max^*(\alpha_0 + \gamma_0, \alpha_1 + \gamma_1)$

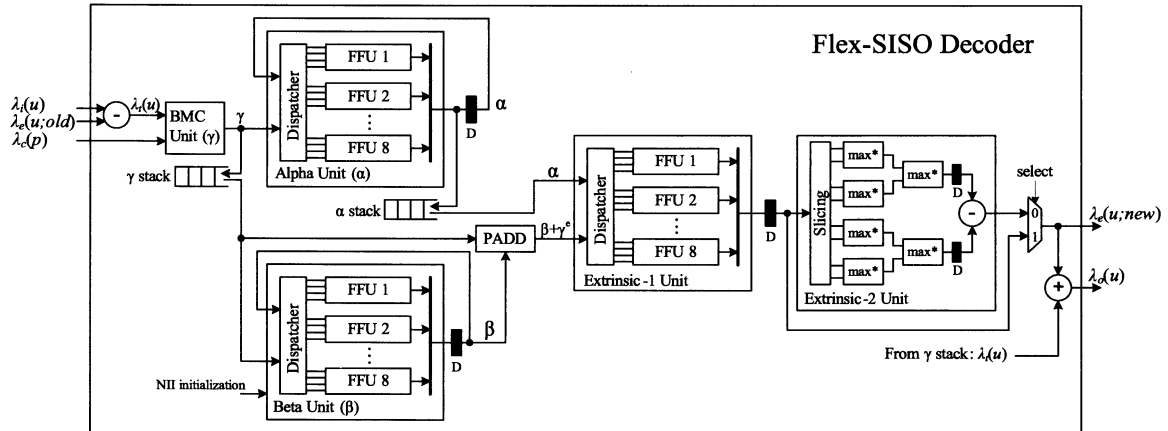


Figure 5.33 : Flexible SISO decoder architecture.

algorithm [108, 127] as described in Chapter 4. The NII approach can avoid the calculation of training sequences as initialization values for the β state metrics, instead the boundary metrics are initialized from the previous iteration. As a result, the decoding latency is smaller than the traditional sliding window algorithm which requires a calculation of training sequences [107, 110], and thus only one β unit is required. Moreover, this solution is very suitable for high code-rate Turbo codes, which require a very long training sequence to obtain reliable boundary state metrics. Note that this scheme would require an additional memory to store the boundary state metrics.

A dataflow graph for the NII sliding window algorithm is depicted in Fig. 5.34, where the X-axis represents the trellis flow and the Y-axis represents the decoding time so that a box may represent the processing of a block of L data in L time steps, where L is the sliding window size. In the decoding process, the α metrics are computed in the natural order whereas the β metrics and the extrinsic LLR (λ_e) are computed in the reverse order. By using multiple FFUs, the α and β units are able to compute the state metrics in parallel, leading to a real time decoding with a latency of L .

The decoder works as follows. The decoder uses the soft LLR value $\lambda_i(u)$ and old extrinsic value $\lambda_e(u; old)$ to compute $\lambda_t(u)$ based on (5.32). A branch metric calculation (BMC) unit is used to compute the branch metrics $\gamma(u, p)$ based on (5.34), where $u, p \in \{0, 1\}$. Then the branch metrics are buffered in a γ stack for backward (β) metric calculation. The α and β metrics are computed using (5.26)(5.27). The

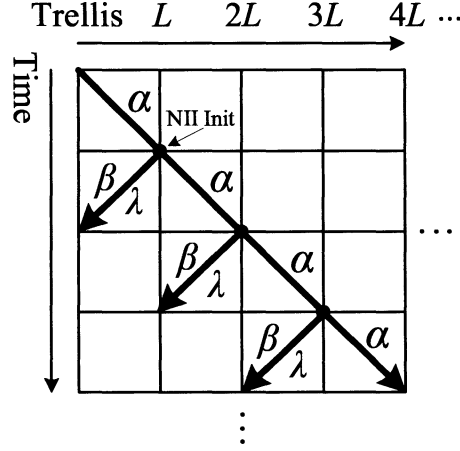


Figure 5.34 : Data flow graph for Turbo decoding.

boundary β metrics are initialized from an NII buffer (not shown in Fig. 5.33). A dispatcher unit is used to dispatch the data to the correct FFUs in the α/β unit. Each α/β unit has fully-parallel FFUs (8 of them), so the 8-state convolutional trellis can be processed at a rate of one-stage per clock cycle.

To compute the extrinsic LLR as defined in eq. (5.25), we first add β metrics with the extrinsic branch metrics $\gamma^e(p)$, where $\gamma^e(p)$ is retrieved from the γ stack, as $\gamma^e(0) = 0$, $\gamma^e(1) = \gamma(0, 1) = \lambda_c(p)$. The extrinsic LLR calculation is separated into two phases which is shown in the right part of Fig. 5.33. In phase 1, the extrinsic-1 unit performs 8 ACSA operations in parallel using 8 FFUs. In phase 2, the extrinsic-2 unit performs 6 $\max^*(a, b)$ operations and 1 subtraction. Finally, the soft LLR $\lambda_o(u)$ is obtained by adding $\lambda_e(u; new)$ with $\lambda_t(u)$, where $\lambda_t(u)$ is also retrieved from the γ stack, as $\lambda_t(u) = \gamma(1, 0)$.

In the LDPC mode, a substantial subset (more than 90%) of the logic gates will be reused from the Turbo mode. As shown in Fig. 5.35, three major functional units (α unit, β unit, and the extrinsic-1 unit) and two stack memories are reused in the LDPC mode. The extrinsic-2 unit will be de-activated in the LDPC mode. The decoder can process 8 single parity check codes in parallel because each of the α unit, β unit, and extrinsic-1 unit has 8 parallel FFUs.

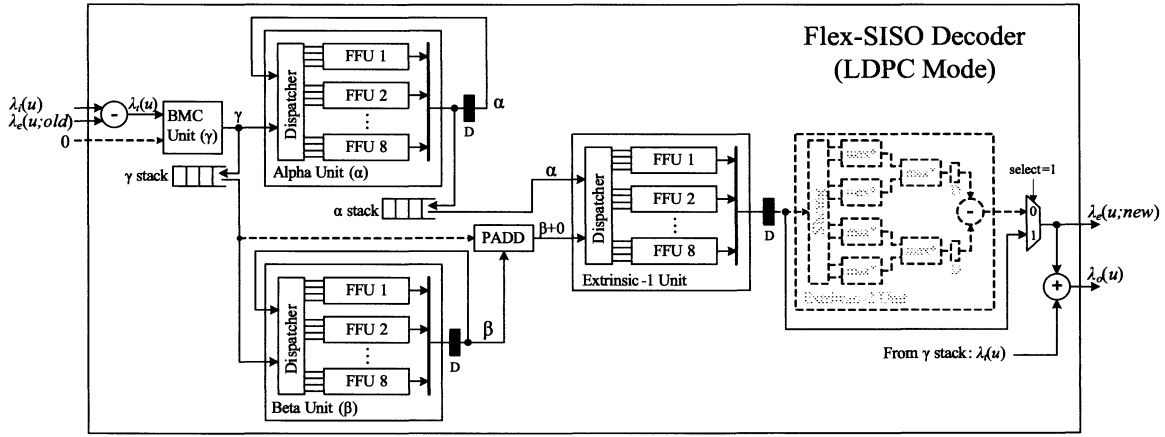


Figure 5.35 : Flexible SISO decoder architecture in LDPC mode.

The dataflow graph of the LDPC decoding (c.f. Fig. 5.29) is very similar to that of the Turbo decoding (c.f. Fig. 5.34). The decoder works as follows. The decoder first computes $\lambda_t(u)$ based on (5.21). In the LDPC mode, the branch metric γ is equal to $\lambda_t(u)$. Prior to decoding, the α and β metrics are initialized to the maximum value. We assume that the check node degree is L . In the first L cycles, the α unit recursively computes the α metrics in the forward direction and stores them in an α stack. In the next L cycles, the β unit recursively computes the β metrics in the backward

direction. At the same time, the extrinsic-1 unit computes the extrinsic LLRs using the α and β metrics. While the β unit and the extrinsic-1 unit are working on the first data stream, the α unit can work on the second stream which leads to a pipelined implementation.

5.8.6 LDPC/Turbo Parallel Decoder Architecture Based on Multiple Flex-SISO Decoders

For high throughput applications, it is necessary to use multiple SISO decoders working in parallel to increase the decoding speed. For parallel Turbo decoding, multiple SISO decoders can be employed by dividing a codeword block into several sub-blocks and then each sub-block is processed separately by a dedicated SISO decoder [112, 113, 114, 103, 12]. For LDPC decoding, the decoder parallelism can be achieved by employing multiple check node processors [17, 65, 66, 67, 76].

Based on the Flex-SISO decoder core, we propose a parallel LDPC/Turbo decoder architecture which is shown in Fig. 5.36. As depicted, the parallel decoder comprises P Flex-SISO decoder cores. In this architecture, there are three types of storage. Extrinsic memory (Ext-Mem) is used for storing the extrinsic LLR values produced by each SISO core. APP memory (APP-Mem) is used to store the initial and updated LLR values. The APP memory is partitioned into multiple banks to allow parallel data transfer. The Turbo parity memory is used to store the channel LLR values for each parity bit in a Turbo codeword. This memory is not used for LDPC de-

coding (parity bits are treated as information bits for LDPC decoding). Finally, two permuters are used to perform the permutation of the APP values back and forth.

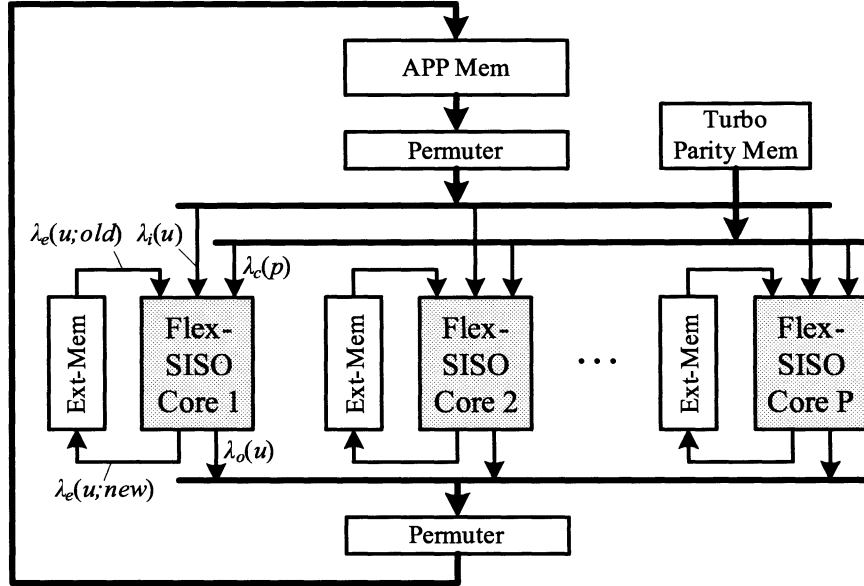


Figure 5.36 : Parallel LDPC/Turbo decoder architecture based on multiple Flex-SISO decoder cores.

5.9 Summary

In this chapter, we have presented high-throughput LDPC decoder architectures for QC-LDPC codes. We propose a multi-layer parallel LDPC decoding algorithm and describe a multi-layer LDPC decoder architecture to achieve 3 Gbps decoding speed. To support both LDPC and Turbo codes, we propose a unified decoder architecture which can be dynamically configured for both codes with a small hardware overhead, based on combining some of the architecture concepts from Chapter 4 on Turbo decoding with the current chapter on LDPC decoding.

Chapter 6

ASIC and FPGA Implementation Results

In this chapter, we present the ASIC (application-specific integrated circuit) and FPGA (field-programmable gate array) implementation results of various MIMO detectors and channel decoders. The algorithms and architectures were presented in Chapters 3, 4, and 5, with Chapter 3 focusing on MIMO detection, Chapter 4 focusing on Turbo decoders, and Chapter 5 focusing on LDPC and joint LDPC/Turbo decoders. First, we will present results on our Rice WARP testbed which is an efficient verification environment before the creation of a VLSI ASIC acceleration design.

6.1 Decoder Accelerator Design for WARP Testbed

We have implemented a channel decoder accelerator for the Rice WARP Wireless Research Platform [128, 129]. The Rice Wireless Research Platform is reconfigurable and consists of DSP and FPGA devices along with RF radios and high speed AD and DA converters. Experiments on the testbed can be performed to allow for algorithm and partitioning verification, identification of unforeseen bottlenecks, and over the air bit and frame error rate determination. The programmable transceiver hardware is connected to a general purpose host computer for control and interfacing. The testbed platform currently utilizes Mathworks Simulink environments for coordination and

execution scheduling. Wireless algorithm design and mapping to parallel architecture prototypes on the FPGA boards is done via the Xilinx System Generator design tools. Additional modules can be created in Verilog HDL and either synthesized for ASIC analysis or mapped to FPGA for inclusion in the Xilinx System Generator design flow. The testbed uses the custom WARP board with Xilinx Virtex-II Pro and Virtex 4 FPGA devices. WARP allows for rapid prototyping with the integrated Maxim/Sharp 2.4 GHz radio unit daughtercards for end-to-end laboratory experiments. Fig. 6.1 shows the block diagram of the WARP testbed.

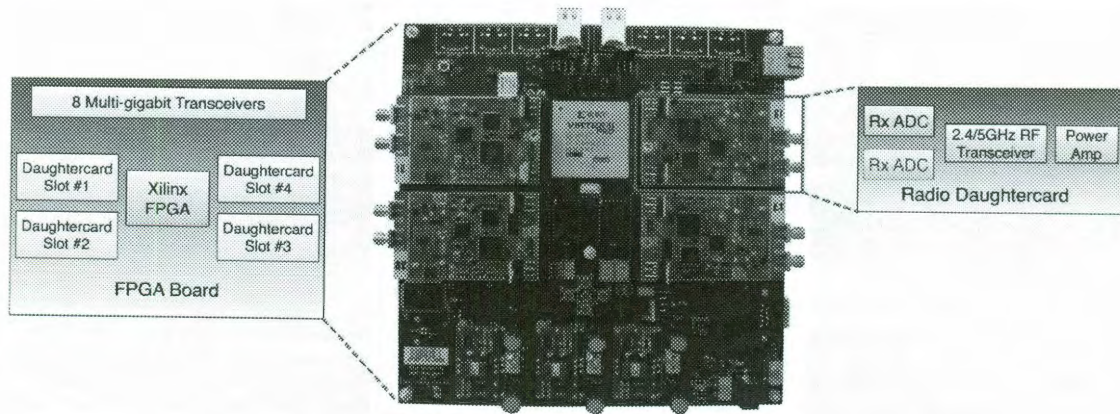


Figure 6.1 : WARP testbed, including the custom Xilinx FPGA board and the radio daughtercards.

We have implemented an FEC codec (convolutional encoder + Viterbi decoder) for the WARP OFDM reference design (<http://warp.rice.edu/trac/wiki/OFDMReferenceDesign>). The most recent version of the OFDM reference design is v15.0. All of the PHY components are open-source and are available in the repository (with revision 1580 for FPGA v1 and svn revision 1585 for FPGA v2).

The design is built using the 10.1 release of the Xilinx tools (ISE 10.1.03 + IP3, Sysgen 10.1.3.1386). In this design, a $K=7$ convolutional code is used. The code structure and the puncture pattern are compliant with the IEEE 802.11a standard.

The FEC codec supports all three modes of the current WARP OFDM PHY: 1) SISO mode, 2) 2×2 MIMO mode, and 3) 2×2 or 2×1 Alamouti mode. The FEC codec supports three modulation types: 1) BPSK, 2) QPSK, and 3) 16-QAM. The coding can be turned on and off by programming the control register. The coding rate can be changed by modifying the second byte of the packet header. Four different code rates are supported: $1/2$, $2/3$, $3/4$, and 1.

The FEC encoder was implemented with Verilog and was integrated into the Sysgen model as a black-box, which is a standard port to include alternate HDL blocks. Fig. 6.2 shows the connection between the encoder and the rest of the Sysgen blocks. As can be seen, the encoder sits between the “data_buffer” block and the “PktBuffer_CRC1” block. The encoder will pre-fetch the data (scrambled information data) from the “PktBuffer_CRC1” block and encode it. The encoded bits are stored into a local small buffer. When this buffer is full, the encoder will stop fetching data from the “PktBuffer_CRC1” block. When the encoder sees a new data byte request from the “data_buffer” block, it will return a coded data byte to the “data_buffer” block. When the coding is turned off, the encoder will bypass the scrambled information data to the “data_buffer” block.

The FEC decoder was also implemented with Verilog and is integrated into Sysgen

as a black-box. Fig. 6.3 shows the connection between the FEC decoder and the other Sysgen blocks. The FEC decoder takes I and Q data and produce the decoded data in bytes. The decoded data are then sent to the "Data Buffer" block for further processing, e.g. CRC error checking.

The FEC codec takes about 12% of the slices in the Virtex-2 Pro FPGA device. The Verilog codes will be uploaded to the repository once they are fully tested. The FEC encoder and decoder support real-time encoding and decoding with a very low latency (the encoder has zero latency and the decoder has less than 50 clock cycles latency).

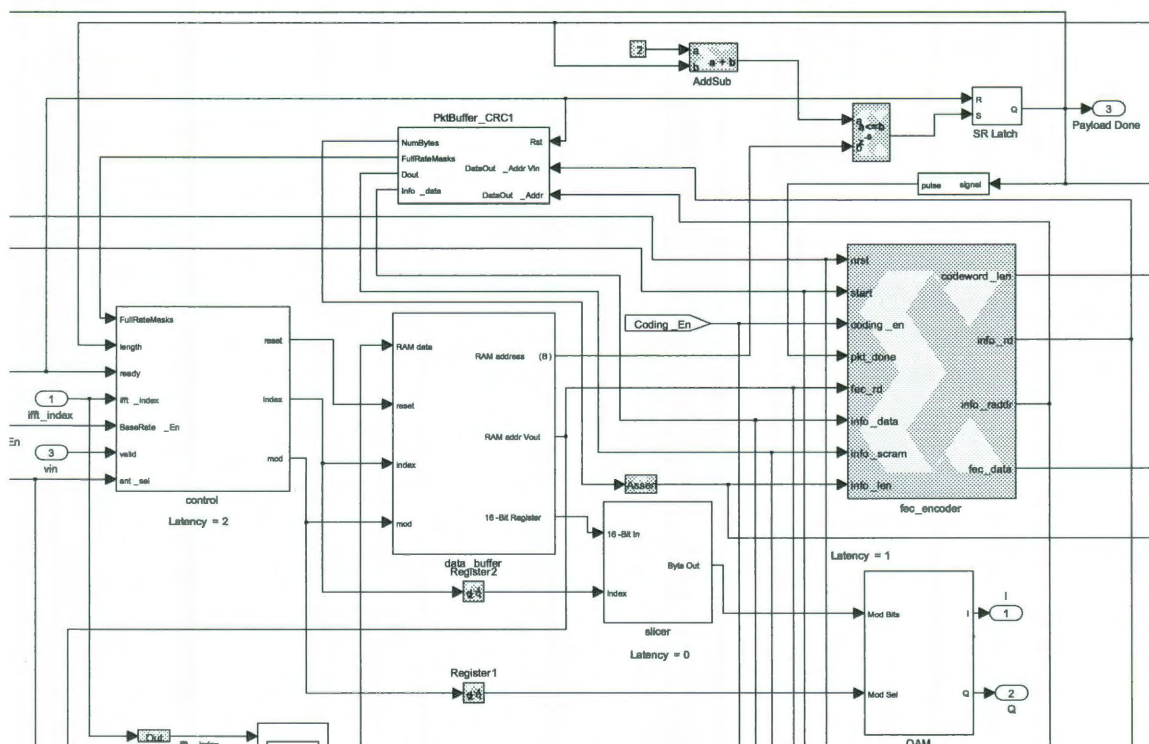


Figure 6.2 : FEC encoder (verilog black-box) integration with WARP MIMO-OFDM System Generator model.

Figure 6.3 : FEC decoder (verilog black-box) integration with WARP MIMO-OFDM System Generator model.

6.2 VLSI Implementation Results for MIMO Detectors

6.2.1 Trellis-Search MIMO Detector, $M = 1$

In chapter 3, we have described the VLSI architectures for the trellis-search MIMO detectors. To evaluate the hardware complexity of the proposed MIMO detector architecture, we implemented a $M = 1$ trellis-search MIMO detector (cf. Section 3.1) using Verilog HDL [6, 7, 8]. To save area, this detector is based on the folded architecture as described in Chapter 3.

This 4×4 16-QAM soft MIMO detector has been synthesized (using Synopsys Design Compiler), placed and routed (using Cadence SoC Encounter) for a TSMC 65nm CMOS technology. Figure 6.4 shows the VLSI layout view of the MIMO detector. The fixed-point bit precision for \mathbf{R} and $\hat{\mathbf{y}}$ are 10 bits. The LLR outputs are represented in 7 bits. Based on the fixed-point simulation results, the finite word-length implementation leads to negligible performance degradation (about 0.1dB) from using the floating-point representation. The maximum achievable clock frequency is 450 MHz based on the post-layout simulation. The corresponding maximum throughput is 600 Mbps.

Table 6.1 compares the detection throughput and hardware complexity of the proposed detector versus two state-of-the-art detectors from the literature: depth-first soft sphere detector with 256 search operations from [28], and soft K-best detector from [39]. In [39], a real QR decomposition is used with a small $K=5$. Compared to solutions [39, 28], our solution can achieve a faster throughput because we avoid the

Table 6.1 : Architecture comparison with existing MIMO detectors

	Garrett [28]	Guo [39]	This work
Algorithm	Depth-First	K-Best	PPTS ($M = 1$)
Configuration	4×4 16-QAM	4×4 16-QAM	4×4 16-QAM
Throughput	38.8 Mbps	106 Mbps	600 Mbps
Core Area	10 mm^2	0.56 mm^2	0.79 mm^2
Gate Count	1100 K	97 K	550 K
Max Frequency	122.88 MHz	200 MHz	450 MHz
Technology	180 nm	130 nm	65 nm
$\frac{\text{Gates (KG)}}{\text{Throughput (Mbps)}}$	28.4	0.92	0.91

trellis-search MIMO detector, instead of working on the scaled s_k signal, we scale each element in the \mathbf{R} matrix by $\frac{1}{\sqrt{10M_t}} = \frac{1}{\sqrt{40}}$ and use the original QAM symbol s_k in the computation. We use the notation $Q[QI].[QF]$ to represent a fixed point number with QI number of integer bits and QF number of fractional bits so that the total word length is $QI + QF$. Table 6.2 summarizes the fixed point design parameters for the scaled R , received \hat{y} , PED, and LLR, where the PED is rounded to 10 bits between computational blocks. This fixed-point detector has about 0.1 dB performance loss compared to the floating-point detector.

Table 6.2 : Fixed point design parameters for the 4×4 16-QAM MIMO system

Signal	Scaled R	Received \hat{y}	PED	LLR
$Q[QI].[QF]$	$Q1.9$ signed	$Q4.6$ signed	$Q4.6$ unsigned	$Q4.2$ signed

ASIC Implementation Result and Architecture Comparison

As a proof of concept, we have implemented a systolic trellis-search MIMO detector with $M = 2$, and a folded trellis-search MIMO detectors with $M = 2$ for a 4×4 16-QAM system. The two detectors have been described using Verilog HDL, and have been synthesized for a 1.08V TSMC 65nm CMOS technology using Synopsys Design Compiler. Fig. 6.5 shows the VLSI layout view of the systolic detector.

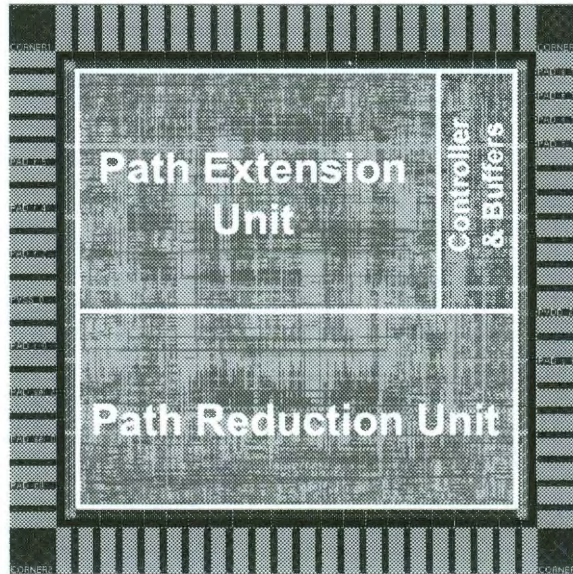


Figure 6.5 : VLSI layout view of the systolic trellis-search MIMO detector ($M = 2$).

Table 6.3 compares the throughput and the hardware complexity of the proposed detectors with two independent works from the literature: a more recent work on depth-first soft sphere detector from [33], and a soft K -Best detector from [39]. Table 6.4 compares the proposed detectors with two related works in our group and our collaborator: a bounded soft sphere detector (BSSD) from [86], and a modified metric

first soft sphere detector (MMF-SSD) from [87].

Since these designs have different technologies, i.e. 65nm, 130nm, 180nm, and 250nm. For a fair comparison, we need to scale these designs into a same technology, i.e. 65nm. To compare silicon area cost, a fair metric is the gate equivalent or gate count, which does not change much as technology node changes. To further compare area efficiency, we define an area efficiency metric (KGate/bit) as:

$$\text{Area efficiency} = \frac{\text{Gate count} \times \text{Frequency}}{\text{Throughput}}. \quad (6.1)$$

This metric does not change much as the technology node changes, and can be used to measure the area efficiency of the design. Similarly, to compare power efficiency, we define an energy efficiency metric (nJ/bit) as:

$$\text{Energy efficiency} = \frac{\text{Normalized power}}{\text{Throughput}}. \quad (6.2)$$

In the equation above, the normalized power is the power number that is scaled to a same technology node, i.e. 65nm, as:

$$\text{Normalized power} = \frac{\text{Power}}{\text{technology scaling factor}^2}. \quad (6.3)$$

As can be seen, the proposed detectors achieve very high data throughput while still maintaining a low area and energy requirement.

In terms of error performance, the proposed trellis detector with $M = 2$ outperforms the K-Best detector with $K = 64$ (cf. Fig. 3.6). Although the depth-first detector with un-limited search steps achieves near-optimal performance, in a practical design, the search steps will be limited to meet the throughput requirement.

However, with limited search steps, the error performance of a depth-first detector quickly degrades. For example, the depth-first MMF-SSD detector from [87] shows a 0.6-0.8 dB performance loss compared to the optimal case.

The trellis MIMO detector with $M = 2$ achieves a balanced tradeoff between hardware complexity and error performance (< 0.3 dB loss). Therefore, the proposed detector is a good solution for the Gbps MIMO detection problem as it achieves both high throughput performance and good error performance.

Table 6.3 : Architecture comparison with two independent works

Reference	Studer [33]	Guo [39]	Systolic	Folded
Algorithm	Depth-First	K -Best, $K=5$	Trellis, $M=2$	Trellis, $M=2$
Configuration	4x4 16-QAM	4x4 16-QAM	4x4 16-QAM	4x4 16-QAM
Clock Frequency	71 MHz	200 MHz	400 MHz	400 MHz
Technology	250 nm	130 nm	65 nm	65 nm
Throughput	10-95 Mbps	106 Mbps	6.4 Gbps	2.1 Gbps
Core Area	1.9 mm ²	0.56 mm ²	3.19 mm ²	1.18 mm ²
Gate Count	56.8 K	97 K	2.22 M	820 K
Power	N/A	N/A	210 mW	81 mW
Area Efficiency	403-42	183	138	156
Energy Efficiency	N/A	N/A	0.03	0.04

Table 6.4 : Architecture comparison with two internal works

Reference	Radosav. [86]	Myllyla [87]	Systolic	Folded
Algorithm	BSSD	MMF-SSD	Trellis, $M=2$	Trellis, $M=2$
Configuration	4x4 16-QAM	4x4 16-QAM	4x4 16-QAM	4x4 16-QAM
Clock Frequency	200 MHz	250 MHz	400 MHz	400 MHz
Technology	130 nm	180 nm	65 nm	65 nm
Throughput	72 Mbps	31-121 Mbps	6.4 Gbps	2.1 Gbps
Core Area	0.57 mm ²	0.59 mm ²	3.19 mm ²	1.18 mm ²
Gate Count	210 K	43.9 K	2.22 M	820 K
Power	43.45 mW	83 mW	210 mW	81 mW
Area Efficiency	583	354-90	138	156
Energy Efficiency	0.15	0.09	0.03	0.04

6.3 VLSI Implementation Results for LTE Turbo Decoders

6.3.1 Highly-Parallel LTE-Advanced Turbo Decoder

A highly-parallel 3GPP LTE/LTE-Advanced Turbo decoder, which consists of 64 Radix-2 SW-MAP decoder cores (cf. Chapter 4 Section 4.4), has been synthesized, placed and routed for a 1.0V 8-metal layer TSMC 65nm CMOS technology [11]. The decoder has scalable parallelism. The decoder can employ 64, 32, and 16 MAP units when the block size $N \geq 2048$, $N \geq 1024$, and $N \geq 512$, respectively. For small block size $N < 496$, the decoder can use up to 8 MAP cores. Figure 6.6 shows the top layout view of this ASIC which shows the core area of this decoder. The fixed-point bit precisions are as follows: the channel symbol LLRs for systematic and parity

bits are represented with 6-bit signed numbers (with 2 fractional bits), the internal α and β state metrics are represented with 10-bit unsigned integer numbers (modulo normalization), and the extrinsic LLRs are represented with 8-bit signed integer numbers. Based on the fixed-point simulation result, the finite word-length implementation leads to negligible BER performance degradation from using the floating-point representation. The maximum achievable clock frequency is 400 MHz based on the post-layout simulation. The corresponding maximum throughput is 1.28 Gbps (at 6 iterations) with a core area of 8.3 mm².

We compare the proposed Turbo decoder with existing Turbo decoders from [112], [113], [58], and [61]. In [112], a parallel Turbo decoder based on 7 MAP decoders is presented. In order to avoid memory contention, a custom designed interleaver, which is not standard compliant, is used. In [113], a 3G-compliant parallel Turbo decoder based on the row-column permutation interleaver is introduced. In [58], a 188-mode Turbo decoder chip for 3GPP LTE standard is presented. In this decoder, 8 MAP units are used to achieve a maximum decoding throughput of 129Mbps (at 8 iterations). In [61], a Radix-4 Turbo decoder is proposed for 3GPP LTE and WiMax standards. A maximum throughput of 186Mbps is supported by employing 8 MAP units (at 8 iterations). Table 6.5 summarizes the implementation results of the proposed decoder and the hardware comparison with existing decoders. As can be seen, the proposed decoder supports the 3GPP LTE-Advanced throughput requirement (1 Gbps) at a small area cost, and achieves a good energy efficiency.

Table 6.5 : Turbo decoder ASIC comparison

	This work [11]	Bougard [112]	Thul [113]	Wong [58]	Kim [61]
Max. block size	6144	432	5120	6144	6144
MAP cores	64	7	6	8	8
Maximum iterations	6	6	6	8	8
Technology	65nm	180nm	180nm	90nm	130nm
Supply voltage	0.9V	1.8V	NA	1.0V	1.2V
Clock frequency	400MHz	160MHz	166MHz	275MHz	250MHz
Core area	8.3mm ²	7.16mm ²	13mm ²	2.1mm ²	10.7mm ²
Gate Equivalent	5.8M	587K [†]	1.3M [‡]	740K ^{††}	800K
Arithmetic Logic	4.9M	373K	N/A	N/A	500K
Throughput	1.28Gbps	75.6Mbps	60Mbps	129Mbps	186Mbps
Power consumption	845mW	N/A	N/A	219mW	N/A
Energy efficiency (nJ/bit/iteration)	0.11	1.45	1.65	0.21	0.61

[†] The gate count is estimated based on the chip data in this thesis.

[‡] The unit cell area is assumed to be 10.00 μm^2 for 180nm technology.

^{††} The unit cell area is assumed to be 2.82 μm^2 for 90nm technology.

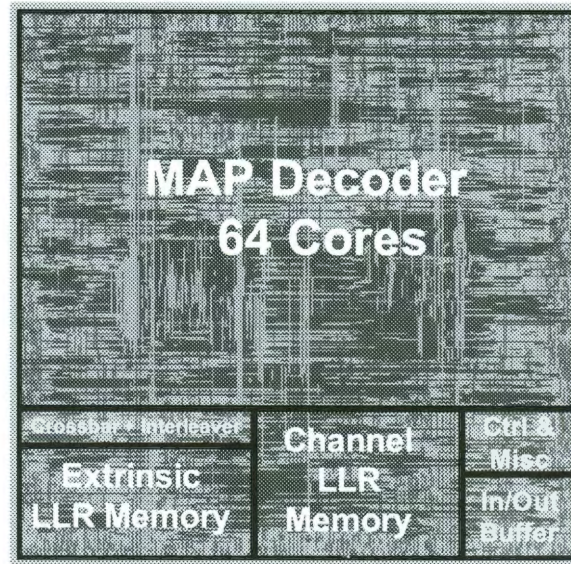


Figure 6.6 : VLSI layout view of an LTE-advanced Turbo decoder.

6.4 VLSI Implementation Results for LDPC Decoders

6.4.1 IEEE 802.11n LDPC Decoder

An IEEE 802.11n LDPC decoder is implemented based on the single-layered offset min-sum algorithm [18]. The decoder was implemented in Verilog HDL and synthesized on a TSMC $0.13\mu m$ standard cell library. Table 6.6 shows a summary of synthesis results. Complexity is measured in equivalent gates for logic and in bits for memories. An overall complexity of 90 K logic gates is measured for the non-pipelined implementation, plus 77,760 bits of RAM. In comparison, 195 K logic gates is measured for the pipelined implementation, plus 77,760 bits for memories based on the additional register and control needed for pipelined operation.

A Verilog RTL simulation model was used to measure average throughput v.s.

SNR level. For instance, at a rather low SNR of 1.0 dB, the pipelined decoder can achieve 150 Mbps. While at a higher SNR of 2.2 dB, the pipelined decoder can achieve about 1 Gbps.

Table 6.6 : IEEE 802.11n LDPC decoder design statistics [18].

	Non-pipelined	Pipelined
Frequency	400 MHz	400 MHz
Area	1.3 mm^2	1.9 mm^2
Logic gates	90 K	195 K
Total memory	77,760 bits	77,760 bits
Throughput@2.2dB SNR	500 Mbps	1 Gbps
Throughput@1.0dB SNR	80 Mbps	150 Mbps

6.4.2 Variable Block-Size and Multi-Rate LDPC Decoder

A flexible LDPC decoder which supports variable block sizes from 360 to 4200 bits in fine steps, where the step size can be 24 (at rate $1/4$, $1/3$, $1/2$, $2/3$, $3/4$, $5/6$ and $7/8$), or 25 (at rate $2/5$, $3/5$ and $4/5$), or 27 (at rate $8/9$), or 30 (at rate $9/10$), was described in Verilog HDL [17]. Layout was generated for a TSMC $0.13\mu m$ CMOS technology as shown in Fig. 6.7. Table 6.8 compares this decoder with two existing LDPC decoders from [69] and [80].

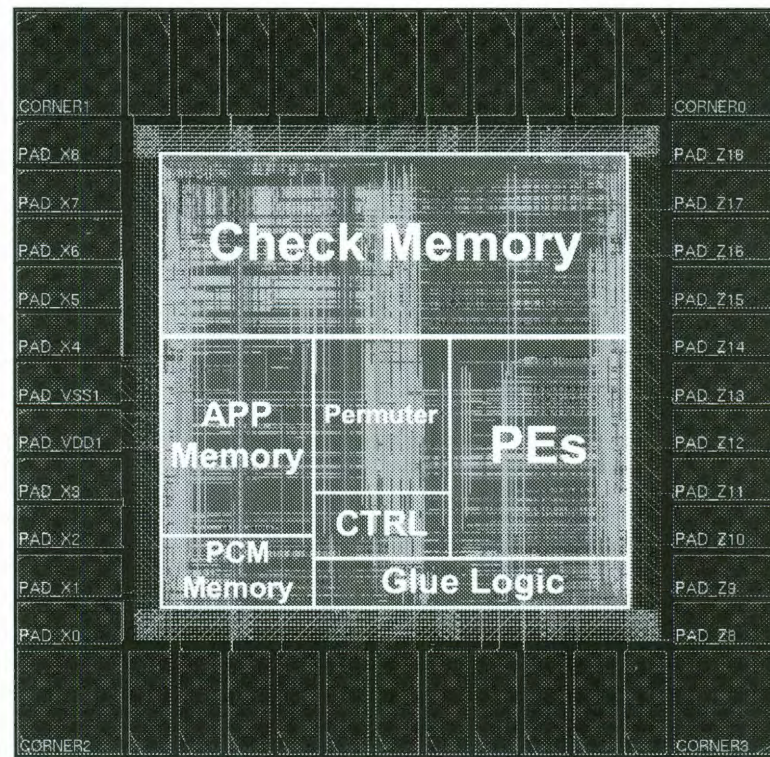


Figure 6.7 : VLSI layout view for a variable block-size and multi-rate LDPC decoder.

Table 6.7 : Variable-size LDPC decoder comparisons

	This work [17]	Blanksby [69]	Mansour [80]
Throughput	1.0 Gbps@2.2dB	1.0 Gbps	1.3Gbps@2.2dB
Area	4.5 mm^2	52.5 mm^2	14.3 mm^2
Frequency	350 MHz	64 MHz	125 MHz
Power	740 mW	690 mW	787 mW
Block size	360 to 4200 bit	1024 bit fixed	2048 bit fixed
Code Rate	1/4 : 9/10	1/2 fixed	1/16 : 14/16
Technology	0.13 μm , 1.2V	0.16 μm , 1.5V	0.18 μm , 1.8V

6.4.3 An IEEE 802.11n/802.16e Multi-Mode LDPC Decoder

In order to support even more wireless systems than our result in Section 6.4.2, a multi-mode LDPC decoder which supports both IEEE 802.11n and IEEE 802.16e has been synthesized on a TSMC 90nm 1.0V 8-metal layer CMOS technology [16]. The detailed VLSI architecture has been described in Chapter 5 Section 5.5. Fig. 6.8 shows the VLSI layout view of the LDPC decoder. Table 6.8 compares this decoder with the state-of-the-art LDPC decoders of [130] and [80]. The decoder in [130] has the flexibility to support 19 modes of LDPC codes in the WiMax standard, however it will not support the higher data rates envisioned for 4G and IMT-Advanced. The decoder in [80] has a throughput of 640 Mbps, but it does not have the flexibility to support multiple codes. As can be seen, our decoder shows significant performance improvement in throughput, flexibility, area and power.

Table 6.8 : IEEE 802.11n/802.16e LDPC decoder comparison

	This Work [16]	Shih [130]	Mansour [80]
Flexibility	802.16e/.11n	802.16e	2048-bit fixed
Max Throughput	1 Gbps	111 Mbps	640 Mbps
Total Area	3.5 mm^2	8.29 mm^2	14.3 mm^2
Max Frequency	450 MHz	83 MHz	125 MHz
Peak Power	410 mW	52 mW	787 mW
Technology	90 nm	0.13 μm	0.18 μm
Max Iteration	10	8	10
Algorithm	Full BP	Min-Sum	Linear Apprx.

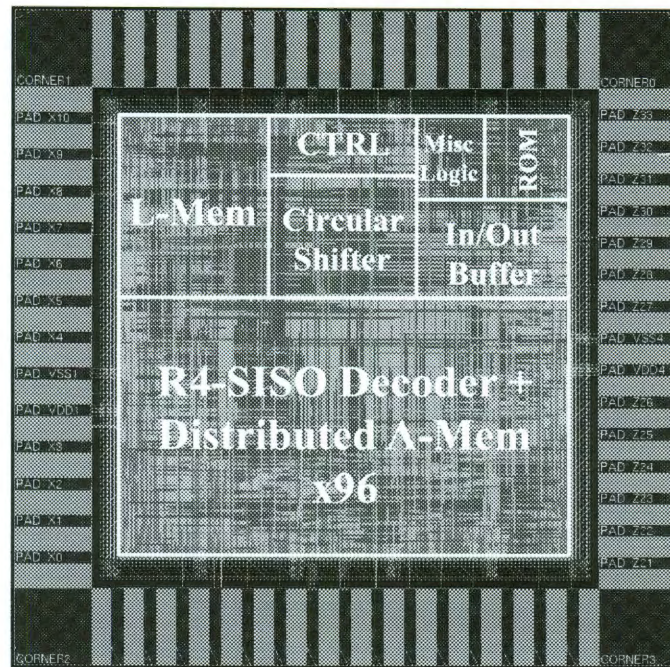


Figure 6.8 : VLSI layout view of an IEEE 802.11n/802.16e multi-mode LDPC decoder.

As low power design is critical for wireless receivers, in order to save power, we have implemented a simple and effective early termination criteria for stopping the iteration process. The decoding will stop if the following two conditions are satisfied: 1) the hard decisions for the information bits based on their LLR values do not change over two successive iterations, and 2) the minimum of the absolute values of the information bit LLRs is larger than a pre-defined threshold. Fig. 6.9 (a) shows the power consumption for different SNR levels for a block size of 2304 bits LDPC code with a maximum iteration number of 10. As shown in Fig. 6.9 (a), when the wireless channel is good, the decoding needs fewer iterations to converge, which therefore saves substantial power (up to 65% power reduction). Another power saving technique is

to use distributed SISO decoders and memory banks. Fig. 6.9 (b) shows the power reduction from deactivating the unused SISO decoders and memory banks when the LDPC code size is small.

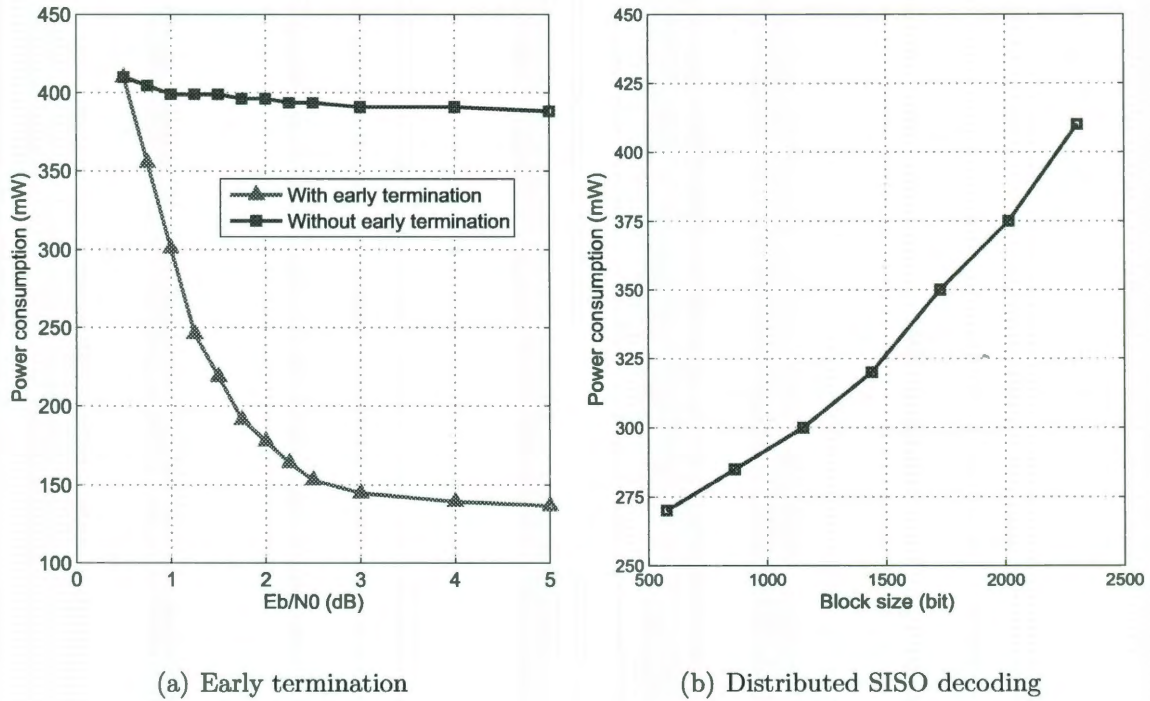


Figure 6.9 : Two power reduction techniques

6.4.4 LDPC Decoder Implementation Using High Level Synthesis Tool

Because of design complexity and variation needed as shown in the thesis, there is much research interest in using high level synthesis (HLS) tools to design LDPC decoders. High level synthesis maps from C/C++ codes to Verilog/VHDL RTL codes. As a case study, we created a flexible LDPC decoder which fully supports the IEEE 802.16e WiMax standard using a high level synthesis design tool [15], the PICO

[131, 132] tool. The generated RTL was synthesized using Synopsys Design Compiler, and placed & routed using Cadence SoC Encounter on a TSMC 65nm 0.9V 8-metal layer CMOS technology. The VLSI layout view of this decoder with a core area of 1.2 mm^2 (standard cells + SRAMs) is shown in Fig. 6.10. Table 6.9 compares our decoder with the state-of-the-art LDPC decoders of [65] and [66]. A fair comparison is difficult to make because of different design parameters. However, it can be roughly inferred that the PICO-generated decoder can achieve comparable performance with the hand designed decoders in terms of throughput, area, and power.

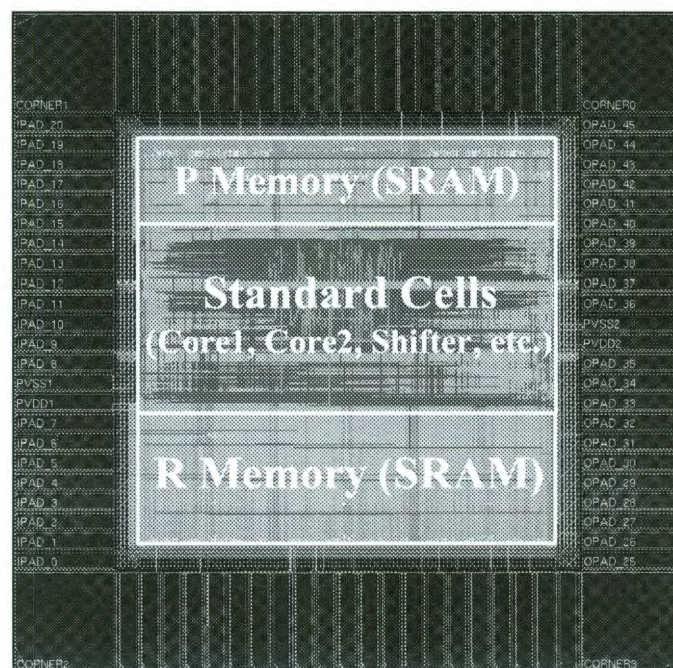


Figure 6.10 : VLSI layout view of the LDPC decoder created from high level synthesis.

The PICO scheduler can analyze the underlying data flow graph, and set those idle registers' "enable" signals to "0" when the module has no activity. PICO also

Table 6.9 : LDPC decoder comparisons, HLS v.s. manual design.

	This Work [15]	Rovini [65]	Brack [66]
Core Area	1.2 mm ²	0.74 mm ²	1.337 mm ²
Max Frequency	400 MHz	240 MHz	400 MHz
Max Power	180 mW	235 mW	NA
Technology	65 nm	65 nm	65 nm
Quantization	6	5	6
Number of Iterations	10	13	25-20
Max Code Length	2304	1944	2304
Memory (SRAM)	82,944 bit	68,256 bit	0.551 mm ²
Max Throughput @ R=1/2	415 Mbps	178 Mbps	333 Mbps
Max Latency @ R=1/2	2.8 μ s	5.75 μ s	6.0 μ s

provides block-level clock gating which shuts off entire processing blocks to minimize power at an architectural level. Table 6.10 compares the power consumption of a (2304, 1/2) pipelined LDPC decoder with and without clock-gating. SpyGlass [133] was used to conduct the gate-level power estimation (not including external SRAMs). From Table 6.10, we can see a 29% reduction of the “sequential internal power” via clock-gating. It should be noted that the power number shown in Table 6.10 is just the standard cell power consumption number.

Table 6.10 : SpyGlass power estimates with and without clock gating

Power	Leakage	Internal	Switching	Total
W/ clock-gating	3.43mW	46.1mW	22.5mW	72.0mW
W/O clock-gating	3.43mW	64.5mW	22.5mW	90.4mW

6.4.5 Multi-Layer Parallel LDPC Decoder for IEEE 802.11n

A flexible double-layer parallel decoder which fully supports IEEE 802.11n LDPC codes was designed in Verilog HDL [134]. The detailed VLSI architecture of this decoder was described in Chapter 5 Section 5.6. The fixed-point design parameters are as follows. The channel input LLR is represented with 6-bit signed numbers with 2 fractional bits. The word lengths of the extrinsic R values and the APP LLR values are 6 bits and 7 bits, respectively. According to the computer simulation, this fixed-point implementation introduces only a performance loss of 0.05 dB compared to the floating-point implementation.

We have synthesized the decoder for a TSMC 45nm CMOS technology. The maximum clock frequency is 815 MHz and the area is 0.81 mm² based on the Synopsys Design Compiler synthesis result. Table 6.11 summarizes the throughput performance of this double-layer parallel decoder for the decoding of IEEE 802.11n LDPC codes at 15 iterations. Table 6.12 compares the implementation result of our decoder with existing 802.11n LDPC decoders from [65, 68, 122]. The solutions from [65, 68, 122] are all based on the conventional single-layer decoding architecture. Compared to those decoders, our pipelined double-layer parallel decoder achieves a much higher throughput at low complexity.

Table 6.11 : Throughput performance of the multi-layer parallel decoder

Block length	Rate 1/2	Rate 2/3	Rate 3/4	Rate 5/6
648 bits	380 Mbps	520 Mbps	760 Mbps	1.0 Gbps
1296 bits	750 Mbps	1.1 Gbps	1.3 Gbps	2.0 Gbps
1944 bits	1.1 Gbps	1.7 Gbps	2.2 Gbps	3.0 Gbps

Table 6.12 : LDPC decoder comparison for IEEE 802.11n

	This work [134]	Rovini [65]	Gunnarn [68]	Studer [122]
Technology	45 nm	65 nm	130 nm	180 nm
Area	0.81 mm ²	0.74 mm ²	1.85 mm ²	3.39 mm ²
Frequency	815 MHz	240 MHz	500 MHz	208 MHz
Iter.	15	14	5	5
Throughput	3.0 Gbps	410 Mbps	1.6 Gbps	780 Mbps

6.5 VLSI Implementation Results for LDPC/Turbo Multi-Mode Decoder

To support more wireless standards with both LDPC and Turbo coding schemes, we have implemented a joint LDPC/Turbo decoder. This flexible decoder together with the proposed MIMO detector can provide a solution for the more advanced iterative detection and decoding scheme.

6.5.1 Implementation Results for The Flexible Functional Unit

The flexible functional unit (FFU) introduced in Chapter 5 (cf, Fig. 5.32) was first synthesized. The word lengths for X , Y , V , and W are all 9 bits. To evaluate the area

efficiency of the proposed FFU, we have described the LDPC $f(a, b)$ unit, the Turbo ACSA unit, and the FFU in Verilog HDL, and synthesized them on a TSMC 90nm CMOS technology. The maximum achievable frequency (assuming no clock skews) and the synthesized area at two frequencies (400 MHz and 800 MHz) are summarized in Table 6.13. As can be seen, the proposed flexible functional unit FFU has only about 15% area and timing overhead compared to the dedicated functional units. The area efficiency is achieved because many logic gates can be shared between LDPC and Turbo modes.

Table 6.13 : Synthesis results for different functional units

Functional unit	$ f(a, b) $	ACSA	FFU
Max frequency	920 MHz	885 MHz	815 MHz
Area (400MHz)	1192 μm^2	1263 μm^2	1419 μm^2
Area (800MHz)	1882 μm^2	2086 μm^2	2423 μm^2

6.5.2 Implementation Results for The Flex-SISO Decoder

The Flex-SISO decoder introduced in Chapter 5 (cf, Fig. 5.33) has been synthesized on a TSMC 90nm CMOS technology. Table 6.14 summarizes the area distribution of this decoder. The maximum clock frequency is 500 MHz and the synthesized area is 0.098 mm². The Flex-SISO is a basic building block in a LDPC decoder or a Turbo decoder, and can be reconfigured to process an 8-state trellis for a Turbo code, or 8 check rows for an LDPC code. As the baseline design, a single Flex-SISO decoder can

approximately support 30-40 Mbps (LTE) Turbo decoding, or 40-50 Mbps (802.16e or 802.11n) LDPC decoding. In a parallel processing environment, multiple SISO decoders can be used to increase the throughput.

Table 6.14 : Flex-SISO decoder area distribution.

Unit	Area
α -unit	0.014 mm ²
β -unit	0.014 mm ²
Extrinsic-1 unit	0.014 mm ²
Extrinsic-2 unit	0.004 mm ²
α and γ stack memories	0.045 mm ²
Control logic & others	0.007 mm ²
Total	0.098 mm ²

6.5.3 Implementation Results for The Top-level LDPC/Turbo Decoder

We have designed a high-throughput, flexible LDPC/Turbo decoder to support the following three codes: 1) 802.16e WiMAX LDPC code, 2) 802.11n WLAN LDPC code, and 3) 3GPP-LTE Turbo code [14, 19]. Table 6.15 summarizes the performance and design parameters for this decoder. The number of the Flex-SISO decoders is chosen to be 12.

To evaluate the fixed-point decoding performance, we perform float-point and bit-accurate fixed-point simulations for LDPC and Turbo codes using BPSK modulation over an AWGN channel. As a good trade-off between complexity and performance,

Table 6.15 : Performance of the unified LDPC/Turbo decoder.

Codes	Code size	Parallelism	Quant.	Iter.	Throughput	Latency
LDPC 802.16e	576-2304 b	$z = 24-96$	6.2	15	600 Mbps	1590 cycles
LDPC 802.11n	648-1944 b	$z = 27-81$	6.2	15	500 Mbps	1620 cycles
LTE Turbo	40-6144 b	12	6.2	6	450 Mbps	6822 cycles

we use 6.2 (6 bits in total with 2 fractional bits) quantization scheme for channel LLR inputs for fixed-point LDPC and Turbo decoders.

Fig. 6.11 shows the bit error rate (BER) simulation result for a WiMAX LDPC code with code-rate = $1/2$, and code-length = 2304. The maximum number of iterations is 15. As can be seen from Fig. 6.11, the fixed-point FFU solution has a very small performance degradation ($< 0.05\text{dB}$) at BER level of 10^{-6} compared to the floating point solution. We also plot a BER curve for the scaled minsum solution [63], which is a sub-optimal approximation algorithm without using the look-up tables. As can be seen from the figure, the look-up table based FFU solution can deliver a better decoding performance than the scaled minsum solution. The complexity of adding the look-up tables is relatively small because the word length of the data in the look-up table is only 2-bit (cf. Chapter 5 Table 5.2). Figure 6.12 compares the convergence speed of the single-layered decoding algorithm with the standard two-phase flooding decoding algorithm.

Fig. 6.13 shows the BER simulation result for 3GPP-LTE Turbo codes with block

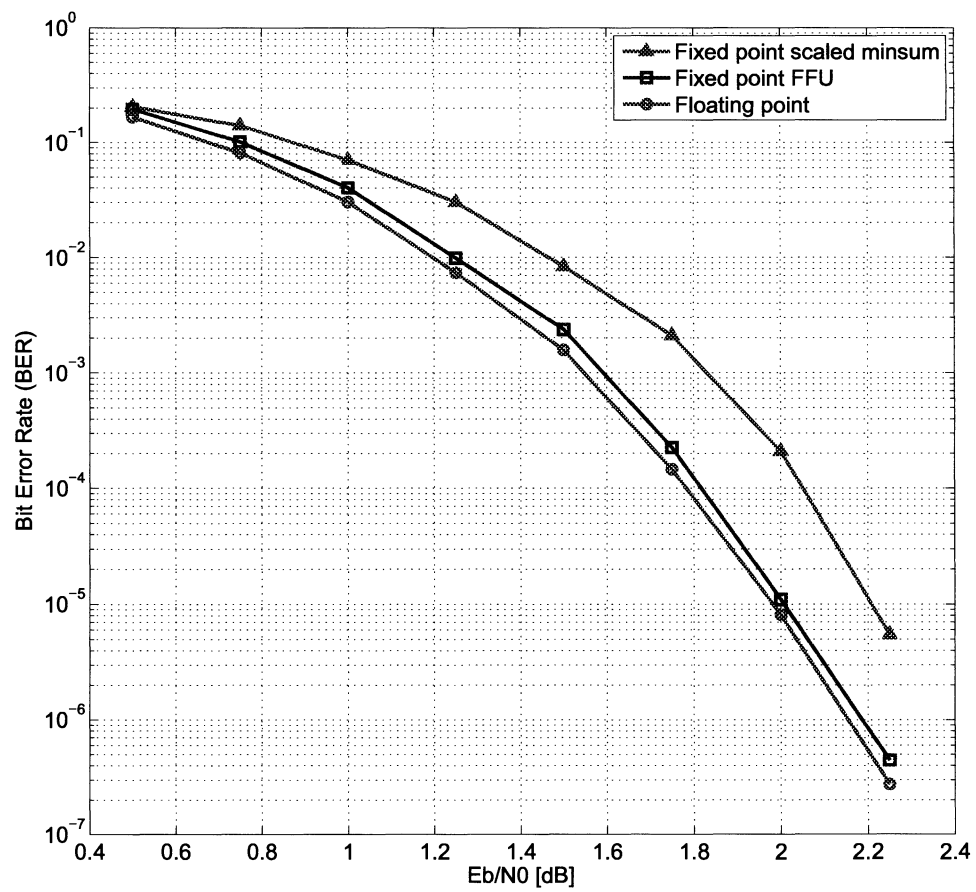


Figure 6.11 : Simulation results for a rate 1/2, length 2304 WiMAX LDPC code.

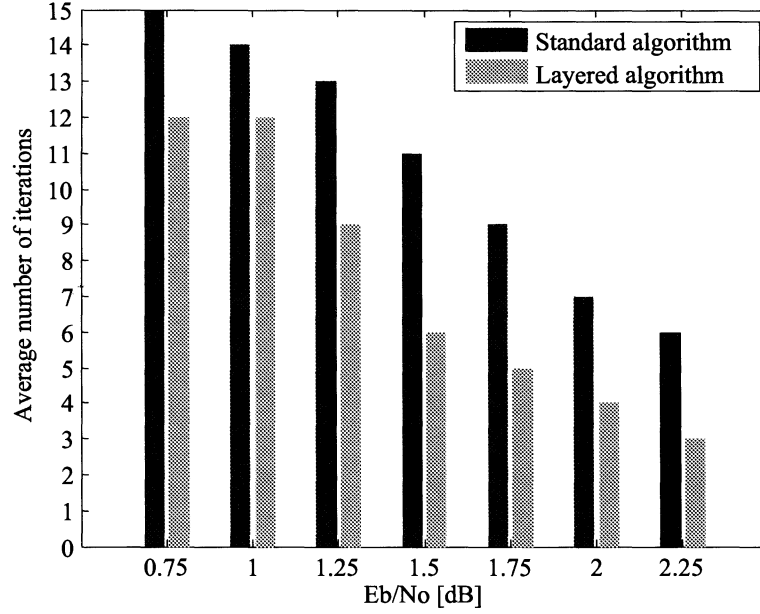


Figure 6.12 : Comparison of the convergence speed.

sizes of 6144, 1024, 240, and 40. The maximum number of Turbo iterations is 6 (12 half iterations). The sliding window length is 32. As can be seen from the figure, the FFU based fixed-point decoder has almost no performance loss compared to the floating point case. The proposed FFU solution will deliver a better decoding performance than the sub-optimal max-logMAP solution.

For LDPC decoding, with 12 available Flex-SISO cores the decoder can process up to $12 \times 8 = 96$ check nodes simultaneously. Because the sub-matrix size z is between 24 to 96 for 802.16e LDPC codes, and 27 to 81 for 802.11n, the proposed decoder always guarantees that all of the z check nodes within a layer can be processed in parallel.

For 3GPP-LTE Turbo decoding, the codeword can be partitioned into M sub-

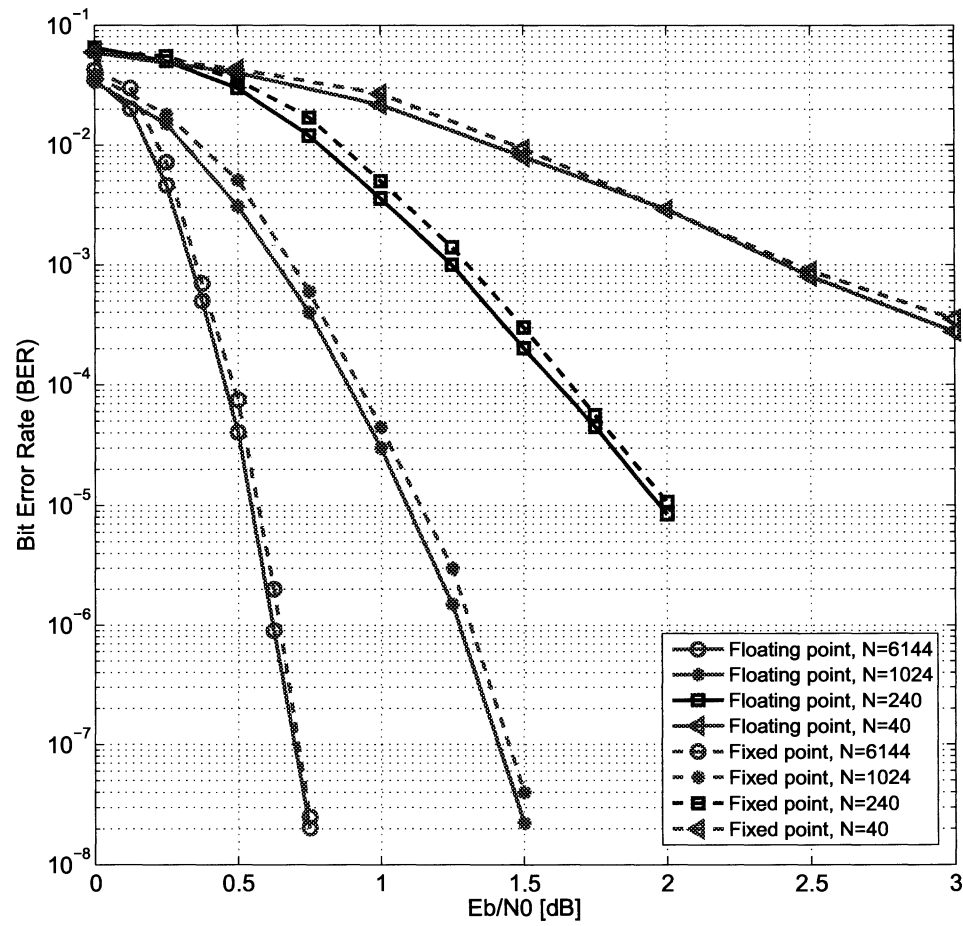


Figure 6.13 : Simulation results for 3GPP-LTE Turbo codes with a variety of block sizes.

blocks for parallel processing. The LTE Turbo code uses a quadratic permutation polynomial (QPP) interleaver [96] so that it allows conflict free memory access as long as M is a factor of the codeword length. There are 188 different codeword sizes defined in LTE. For LTE Turbo codes, all of the codewords can support a parallelism level of 8, some of the codewords can support parallelism levels of 10 or 12. Because we have 12 Flex-SISO cores available, we will dynamically allocate the maximum possible number of Flex-SISO cores ($8 \leq M \leq 12$) constrained on the QPP interleaver parallelism. As an example, for the maximum codeword size of 6144, we can allocate all of the 12 Flex-SISO cores to work in parallel. It should be noted that the parallelism level has some impact on the error performance of the decoder due to the edge effects caused by the sub-block partitioning [135].

This flexible decoder has been implemented in Verilog HDL and synthesized on a TSMC 90nm CMOS technology using Synopsys Design Compiler [14]. The maximum clock frequency of this decoder is 500 MHz. The synthesized core area is 3.2 mm², which includes all of the components in this decoder. Table 6.15 summarizes the features of this decoder. The decoder can be configured to support IEEE 802.16e LDPC codes, IEEE 802.11n LDPC codes, and 3GPP LTE Turbo codes. Compared to a dedicated LDPC decoder solution [16], this flexible decoder has only about 15-20% area overhead when normalized to the same throughput target (with the same number of iterations). Compared to a dedicated Turbo decoder solution [114], our flexible decoder shows only about 10-20% area overhead when normalized to the same

technology and the same throughput and code length. Table 6.5 compares our flexible decoder with existing LDPC/Turbo multi-mode decoder.

Table 6.16 : Architecture comparison with existing flexible LDPC/Turbo solutions.

	This work	[136]	[137]	[138]
Technology	90nm	65nm	130nm	90nm
Clock frequency	500MHz	400MHz	200MHz	NA
Core area	3.2mm ²	0.62mm ²	NA	NA
Throughput (LDPC)	600Mbps	257Mbps	11.2Mbps	70Mbps
Throughput (Turbo)	450Mbps [†]	18.6Mbps [†]	86.5Mbps [‡]	14Mbps [†]

[†] Binary Turbo code.

[‡] Double-binary Turbo code.

6.6 Discussions on the Iterative Receiver Design and Implementation

With the proposed MIMO detector and LDPC/Turbo decoder, an iterative receiver can be realized by connecting the detector to the decoder. For a channel decoder, data buffers would be required because the decoder usually needs to receive a whole codeword block before starting the decoding process. For a MIMO detector, data buffers will also be required because of the channel interleaving. Fig. 6.14 and Fig. 6.15 show the area and power estimation for the iterative receivers for different antennas. In the estimation, we assume each stream is separated coded and multiple LDPC decoders are used for decoding multiple data streams. The detector area and power

for 4 antenna systems are estimated based on the implementation result in Table 6.3, and the decoder area and power for 4 antenna systems are estimated based on the implementation result in Table 6.8. All the numbers are normalized to a same technology, i.e. 65nm. The area and power for 2 and 8 antenna systems are estimated based on the ASIC implementation results for 4 antenna system. Since the streams are separated coded, the decoder complexity increases almost linearly with the number of antennas. However, the detector complexity increases quadratically with the number of antennas, with a complexity of $O((N_t - 1)(N_t - 2)/2)$.

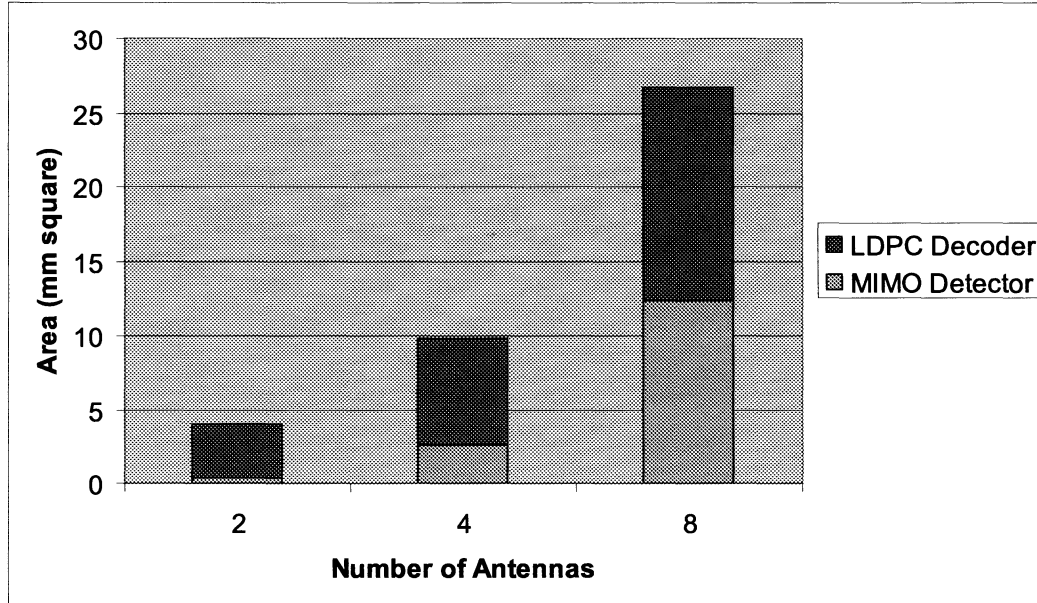


Figure 6.14 : Area estimation for iterative receiver.

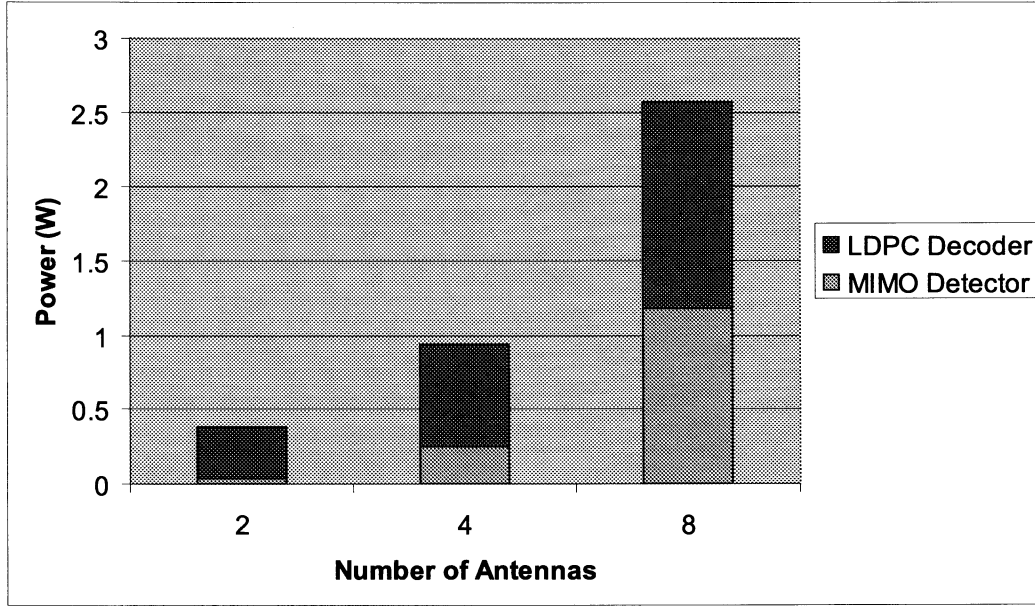


Figure 6.15 : Power estimation for iterative receiver.

6.7 Summary

We have implemented a channel encoder and a channel decoder accelerator for the Rice WARP FPGA testbed. The encoder/decoder was successfully integrated into the WARP MIMO-OFDM System Generator model.

We have implemented various detectors and decoders on ASICs to evaluate the implementation complexity. Compared with the existing detector and decoder solutions, our architecture can achieve a higher throughput performance with reasonable hardware resources.

A potential receiver system for 4G wireless systems could be created from the MIMO detection in Chapter 3 and Chapter 6 connected a channel decoder support-

ing Turbo and LDPC codes from Chapter 4 and Chapter 5. The system could be configured for a single pass or for multiple iterations. Initial simulation results for this architecture were presented in Chapter 3 Section 3.3.

Chapter 7

Conclusion and Future Work

7.1 Conclusion of The Current Results

In this thesis, we introduced a reduced-complexity MIMO detector based on a novel trellis-search algorithm. We represent the search space of the MIMO signal with a trellis structure and convert the MIMO detection problem into a shortest path problem. We proposed a high-throughput VLSI architecture, which can support multiple Gbps data rate. We presented the ASIC implementation results for the proposed MIMO detector architecture. Compared to the existing solutions, the proposed trellis-search based MIMO detector has a significant throughput advantage and a higher area efficiency. The simulation results suggest that the error performance is very close to the optimum MAP detector.

We proposed a parallel Turbo decoding algorithm and architecture to achieve Gbps data rate. We employ multiple MAP decoding units to process a codeword in parallel. By utilizing the contention-free interleaver structure, we avoid the memory conflict problem. We implemented a LTE-Advance Turbo decoder on an ASIC technology.

We proposed a multi-layer parallel LDPC decoding algorithm and architecture to achieve multiple Gbps data rate. The proposed scalable LDPC decoder can be

configured to support different block sizes and code rates. We presented several ASIC implementation results for LDPC decoders for various wireless standards, e.g. IEEE 802.11n and IEEE 802.16e. We further presented a joint LDPC/Turbo decoding algorithm and architecture to support more wireless standards with a small hardware overhead.

We developed an iterative detection and decoding scheme based on the proposed trellis-search detector. In this scheme, the LLR soft values generated by the decoder are fed to the detector, and then the detector restarts a new round of detection to further refine the LLR soft values. The simulation results suggest that a 2.5-3 dB gain can be achieved by such a scheme. The component detector and decoder architectures and ASIC implementations can be combined to create this receiver.

7.2 Future Work

The following issues can be further investigated:

1. Real-value decomposition based trellis-search algorithm: The current trellis-search algorithm is based on the complex-value decomposition of the channel matrix. A variation of this algorithm is to use the real-value decomposition of the channel matrix and to form a real-valued trellis diagram. The number of stages and the number of nodes in each stage will change in a real-valued trellis diagram. It would be an interesting problem to extend the current complex-valued trellis-search algorithm to support real-valued model and compare the complexity and the performance of

these two schemes.

2. Unified decoding architecture: It would be an interesting problem to extend the current joint LDPC/Turbo decoder architecture to support more error-correcting codes such as LDPC convolutional codes, non-binary LDPC codes, and non-binary Turbo codes.

3. Low power design: Next generation CMOS technology would offer more low-power features such as multiple supply voltages and multiple threshold libraries. Furthermore, the 3D CMOS technology is emerging to replace the current planar CMOS technology. The designer can take advantage of these new technologies to reduce the power consumption from all aspects. Low power design is especially useful for hand-held devices, such as cellphones.

Bibliography

- [1] “Evolved Universal Terrestrial Radio Access (EUTRA) and Evolved Universal Terrestrial Radio Access Network (EUTRAN), 3GPP TS 36.300.”
- [2] “General UMTS Architecture, 3GPP TS 23.101 version 7.0.0, June 2007.”
- [3] S. Parkvall, E. Dahlman, A. Furuskar, Y. Jading, M. Olsson, S. Wanstedt, and K. Zangi, “LTE-Advanced - Evolving LTE towards IMT-Advanced,” in *IEEE Vehicular Technology Conference*, Sept. 2008, pp. 1–5.
- [4] G. J. Foschini, “Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas,” *Bell Labs Technical Journal*, vol. 1, no. 2, pp. 41–59, 1996.
- [5] G. Fettweis, T. Hentschel, and E. Zimmermann, “WIGWAM - A Wireless Gigabit System with Advanced Multimedia Support,” in *VDE Kongress, Berlin, Germany*, Oct. 2004, pp. 18–20.
- [6] Y. Sun and J. R. Cavallaro, “High Throughput VLSI Architecture for Soft-Output MIMO Detection Based on a Greedy Graph Algorithm,” in *ACM Great Lakes Symposium on VLSI Design*, May 2009, pp. 445–450.

- [7] —, “Low-complexity and high-performance soft MIMO detection based on distributed M-algorithm through trellis-diagram,” in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, March 2010, pp. 3398–3401.
- [8] —, “A New MIMO Detector Architecture Based on a Forward-Backward Trellis Algorithm,” in *IEEE Asilomar conf. on Signals, Syst. and Computers*, Oct. 2008, pp. 1892–1896.
- [9] Y. Sun, K. Amiri, M. Brogioli, and J. R. Cavallaro, “Application-Specific DSP Accelerators,” *Handbook on Signal Processing Systems (S. Bhattacharyya, E. Deprettere, R. Leupers, J. Takala, Eds.), 1st Edition, Springer, New York, NY*, pp. 329–362, 2010.
- [10] M. Wu, Y. Sun, S. Gupta, and J. R. Cavallaro, “Implementation of a High Throughput Soft MIMO Detector on GPU,” *Journal of Signal Processing Systems (DOI: 10.1007/s11265-010-0523-4, On-Line-First)*, 2010.
- [11] Y. Sun and J. R. Cavallaro, “Efficient Hardware Implementation of A Highly-Parallel 3GPP LTE, LTE-Advance Turbo Decoder,” *Elsevier Integration, the VLSI Journal, Special Issue on Hardware Architectures for Algebra, Cryptology and Number Theory, DOI:10.1016/j.vlsi.2010.07.001, (On-Line)*, July 2010.
- [12] Y. Sun, Y. Zhu, M. Goel, and J. R. Cavallaro, “Configurable and Scalable High Throughput Turbo Decoder Architecture for Multiple 4G Wireless Standards,”

- in *IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, July 2008, pp. 209–214.
- [13] M. Wu, Y. Sun, and J. R. Cavallaro, “Implementation of a 3GPP LTE Turbo Decoder Accelerator on GPU,” in *IEEE Workshop on Signal Processing Systems (SIPS)*, Oct. 2010, pp. 193–198.
- [14] Y. Sun and J. R. Cavallaro, “A Flexible LDPC/Turbo Decoder Architecture,” *Springer Journal of Signal Processing Systems, Special Issue on the 2008 IEEE SiPS Workshop*, DOI: 10.1007/s11265-010-0477-6, (On-Line First), July 2010.
- [15] —, “Scalable and Low Power LDPC Decoder Design Using High Level Algorithmic Synthesis,” in *IEEE International SOC Conference (SoCC)*, Sept. 2009, pp. 267–270.
- [16] —, “A low-power 1-Gbps reconfigurable LDPC decoder design for multiple 4G wireless standards,” in *IEEE International SOC Conference*, Sept. 2008, pp. 367–370.
- [17] Y. Sun, M. Karkooti, and J. R. Cavallaro, “VLSI Decoder Architecture for High Throughput, Variable Block-size and Multi-rate LDPC Codes,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2007, pp. 2104–2107.
- [18] —, “High Throughput, Parallel, Scalable LDPC Encoder/Decoder Architecture for OFDM Systems,” in *IEEE Dallas Circuit and System Workshop on*

- Design, Applications, Integration and Software*, Oct. 2006, pp. 39–42.
- [19] Y. Sun and J. R. Cavallaro, “Unified Decoder Architecture For LDPC/Turbo Codes,” in *IEEE Workshop on Signal Processing Systems (SIPS)*, Oct. 2008, pp. 13–18.
- [20] B. Hochwald and S. Brink, “Achieving Near-Capacity on a Multiple-Antenna Channel,” *IEEE Transactions on Communications*, vol. 51, pp. 389–399, Mar. 2003.
- [21] U. Fincke and M. Pohst, “Improved Methods for Calculating Vectors of Short Length in a Lattice, Including a Complexity Analysis,” *Mathematics of Computation*, vol. 44, no. 170, pp. 463–471, April 1985.
- [22] E. Viterbo and J. Boutros, “A universal lattice code decoder for fading channels,” *IEEE Transactions on Information Theory*, vol. 45, no. 5, pp. 1639–1642, July 1999.
- [23] M. O. Damen, H. E. Gamal, and G. Caire, “On maximum-likelihood detection and the search for the closest lattice point,” *IEEE Transactions on Information Theory*, vol. 49, no. 10, pp. 2389–2402, 2003.
- [24] B. Hassibi and H. Vikalo, “On the sphere-decoding algorithm I. Expected complexity,” *IEEE Transactions on Signal Processing*, vol. 53, no. 8-1, pp. 2806–2818, August 2005.

- [25] H. Vikalo and B. Hassibi, "On the sphere-decoding algorithm II. Generalizations, second-order statistics, and applications to communications," *IEEE Transactions on Signal Processing*, vol. 53, no. 8-1, pp. 2819–2834, August 2005.
- [26] B. Widdup, G. Woodward, and G. Knagge, "A Highly-Parallel VLSI Architecture for a List Sphere Detector," in *IEEE International Conference on Communications*, June 2004, pp. 2720–2725.
- [27] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI Implementation of MIMO Detection Using the Sphere Decoding Algorithm," *IEEE Journal of Solid-State Circuits*, vol. 40, pp. 1566–1577, July 2005.
- [28] D. Garrett, L. Davis, S. ten Brink, B. Hochwald, and G. Knagge, "Silicon Complexity for Maximum Likelihood MIMO Detection Using Spherical Decoding," *IEEE Journal of Solid-State Circuits*, vol. 39, pp. 1544–1552, Sept. 2004.
- [29] Y. Zhang, J. Tang, and K. K. Parhi, "Low Complexity List Updating Circuits for List Sphere Decoders," in *IEEE Workshop on Signal Processing Design and Implementation*, Oct. 2006, pp. 28–33.
- [30] J. Antikainen, P. Salmela, O. Silven, M. Juntti, J. Takala, and M. Myllyla, "Application-specific instruction set processor implementation of list sphere detector," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 3, pp. 1–14, 2007.

- [31] Q. Qi and C. Chakrabarti, "Sphere Decoding for Multiprocessor Architectures," in *IEEE Workshop on Signal Processing Design and Implementation*, Oct. 2007, pp. 50–55.
- [32] X.-M. Huang, C. Liang, and J. Ma, "System Architecture and Implementation of MIMO Sphere Decoders on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 188–197, Feb. 2008.
- [33] C. Studer, A. Burg, and H. Bolcskei, "Soft-output sphere decoding: algorithms and VLSI implementation," *IEEE Journal on Selected Areas in Communications*, vol. 26, pp. 290–300, February 2008.
- [34] M. Myllyla, M. Juntti, and J. R. Cavallaro, "Architecture design and implementation of the increasing radius - List sphere detector algorithm," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 2009.
- [35] J. W. Choi, B. Shim, A. C. Singer, and N. I. Cho, "Low-complexity decoding via reduced dimension maximum-likelihood search," *IEEE Transactions on Signal Processing*, vol. 58, pp. 1780–1793, Mar. 2010.
- [36] K. Wong, C. Tsui, R. Cheng, and W. Mow, "A VLSI architecture of a K-Best lattice decoding algorithm for MIMO channels," in *IEEE International Symposium on Circuits and Systems*, vol. 3, May 2002, pp. 273–276.

- [37] K. Higuchi, H. Kawai, N. Maeda, M. Sawahashi, T. Itoh, Y. Kakura, A. Ushirokawa, and H. Seki, "Likelihood Function for QRM-MLD Suitable for Soft-Decision Turbo Decoding and Its Performance for OFCDM MIMO Multiplexing in Multipath Fading Channel," in *IEEE International Symposium Personal, Indoor and Mobile Radio Communications (PIMRC)*, vol. 2, September 2004, pp. 1142–1148.
- [38] Y. L. C. de Jong and T. J. Willink, "Iterative tree search detection for MIMO wireless systems," *IEEE Transactions on Communications*, vol. 53, no. 6, pp. 930–935, June 2005.
- [39] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-Best sphere decoding for MIMO detection," *IEEE Journal on Selected Areas in Communications*, vol. 24, pp. 491–503, Mar. 2006.
- [40] M. Wenk, M. Zellweger, A. Burg, N. Felber, and W. Fichtner, "K-Best MIMO detection VLSI architectures achieving up to 424 Mbps," in *IEEE International Symposium on Circuits and Systems*, Sept. 2006, pp. 1151–1154.
- [41] Q. Li and Z. Wang, "Improved K-Best sphere decoding algorithms for MIMO systems," in *IEEE International Symposium on Circuits and Systems*, Sept. 2006, pp. 1159–1162.
- [42] S. Chen, T. Zhang, and Y. Xin, "Relaxed K-Best MIMO Signal Detector Design and VLSI Implementation," *IEEE Transactions on Very Large Scale Integration*

- (*VLSI Systems*, vol. 15, pp. 328–337, Mar. 2007.
- [43] M. Shabany, K. Su, and P. G. Gulak, “A pipelined scalable high-throughput implementation of a near-ML K-Best complex lattice decoder,” in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Mar. 2008, pp. 3173–3176.
 - [44] R. Fasthuber et al, “Novel energy-efficient scalable soft-output ssfe mimo detector architectures,” in *Proceedings of the 9th international conference on Systems, architectures, modeling and simulation*, 2009, pp. 165–171.
 - [45] S. Mondal, A. Eltawil, C.-A. Shen, and K. N. Salama, “Design and Implementation of a Sort-Free K-Best Sphere Decoder,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, pp. 1–5, Nov. 2009.
 - [46] T. Cupaiuolo, M. Siti, and A. Tomasoni, “Low-complexity high throughput VLSI architecture of soft-output ML MIMO detector,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Mar. 2010, pp. 1396–1401.
 - [47] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near shannon limit error-correcting coding and decoding: Turbo-codes,” in *IEEE Int. Conf. Commun.*, May 1993, pp. 1064–1070.
 - [48] R. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA MIT Press ed., 1963.

- [49] H. Sadjadpour, N. Sloane, M. Salehi, and G. Nebe, "Interleaver design for turbo codes," *IEEE Journal on Selected Areas in Communications*, vol. 19, pp. 831–837, May 2001.
- [50] D. Garrett, B. Xu, and C. Nicol, "Energy efficient turbo decoding for 3G mobile," in *International symposium on Low power electronics and design*. ACM, 2001, pp. 328–333.
- [51] C. Chaikalis and J. Noras, "Reconfigurable turbo decoding for 3G applications," *Elsevier Signal Processing*, vol. 84, pp. 1957–1972, Oct. 2004.
- [52] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *IEEE Int. Solid-State Circuit Conf. (ISSCC)*, Feb. 2003.
- [53] M. Martina, M. Nicola, and G. Masera, "A Flexible UMTS-WiMax Turbo Decoder Architecture," *IEEE Transactions on Circuits and Systems II*, vol. 55, pp. 369–273, April 2008.
- [54] K. Loo, T. Alukaidey, and S. Jimaa, "High performance parallelised 3GPP turbo decoder," in *IEEE Personal Mobile Communications Conference*, April 2003, pp. 337–342.
- [55] M. Shin and I. Park, "SIMD processor-based turbo decoder supporting multiple third-generation wireless standards," *IEEE Trans. on VLSI*, vol. vol.15, pp. pp.801–810, Jun. 2007.

- [56] Y. Lin, S. Mahlke, T. Mudge, and C. Chakrabarti, "Design and Implementation of Turbo Decoders for Software Defined Radio," in *IEEE SIPS*, Oct. 2006, pp. 22–27.
- [57] P. Salmela, H. Sorokin, and J. Takala, "A Programmable Max-Log-MAP Turbo Decoder Implementation," *HINDAWI VLSI Design*, vol. 2008, pp. 636–640, 2008.
- [58] C.-C. Wong, Y.-Y. Lee, and H.-C. Chang, "A 188-size 2.1mm² reconfigurable turbo decoder chip with parallel architecture for 3GPP LTE system," in *2009 Symposium on VLSI Circuits*, June 2009, pp. 288–289.
- [59] D.-S. Cho, H.-J. Park, and H.-C. Park, "Implementation of an efficient UE decoder for 3G LTE system," in *International Conference on Telecommunications*, June 2008, pp. 1–5.
- [60] J. Berkmann, C. Carbonelli, F. Dietrich, C. Drewes, and W. Xu, "On 3G LTE Terminal Implementation - Standard, Algorithms, Complexities and Challenges," in *International Wireless Communications and Mobile Computing Conference*, Aug. 2008, pp. 970–975.
- [61] J.-H. Kim and I.-C. Park, "A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE," in *IEEE Custom Integrated Circuits Conference*, Sept. 2009, pp. 487–490.

- [62] R. Gallager, “Low-density parity-check codes,” *IEEE Transactions on Information Theory*, vol. 8, pp. 21–28, Jan. 1962.
- [63] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X. Hu, “Reduced-complexity decoding of LDPC codes,” *IEEE Transactions on Communications*, vol. 53, pp. 1288 – 1299, Aug 2005.
- [64] D. Oh and K. Parhi, “Min-Sum Decoder Architectures With Reduced Word Length for LDPC Codes,” *IEEE Transactions on Circuits and Systems I*, vol. 57, no. 1, pp. 105 – 115, Jan. 2010.
- [65] M. Rovini, G. Gentile, F. Rossi, and L. Fanucci, “A Scalable Decoder Architecture for IEEE 802.11n LDPC Codes,” in *IEEE Global Telecommunications Conference*, 2007, pp. 3270–3274.
- [66] T. Brack, M. Alles, T. Lehnigk-Emden, F. Kienle, N. Wehn, N. L’Insalata, F. Rossi, M. Rovini, and L. Fanucci, “Low complexity LDPC code decoders for next generation standards,” in *Design, Automation, and Test in Europe*. ACM, 2007, pp. 331–336.
- [67] H. Zhong and T. Zhang, “Block-LDPC: a practical LDPC coding system design approach,” *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on [see also Circuits and Systems I: Regular Papers, IEEE Transactions on]*, vol. 52, no. 4, pp. 766 – 775, 2005.

- [68] K. Gunnam, G. S. Choi, M. B. Yeary, and M. Atiquzzaman, "VLSI Architectures for Layered Decoding for Irregular LDPC Codes of WiMax," in *IEEE Int. Conf. on Commun.*, June 2007, pp. 4542–4547.
- [69] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 404–412, Mar. 2002.
- [70] M. M. Mansour and N. R. Shanbhag, "High-throughput LDPC decoders," *IEEE Tran. VLSI Syst.*, vol. 11, pp. 976–996, Dec. 2003.
- [71] D. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *IEEE Workshop on Signal Processing Systems (SIPS)*, Oct 2004, pp. 107–112.
- [72] D. Oh and K. Parhi, "Low complexity implementations of sum-product algorithm for decoding low-density parity-check codes," in *IEEE Workshop on Signal Processing Systems (SIPS)*, Oct 2006, pp. 262–267.
- [73] T. Brack, M. Alles, F. Kienle, and N. Wehn, "A Synthesizable IP Core for WIMAX 802.16e LDPC Code Decoding," in *IEEE 17th Int. Symp. Personal, Indoor and Mobile Radio Communications*, 2006, pp. 1 – 5.
- [74] Y. Dai, Z. Yan, and N. Chen, "High-throughput turbo-sum-product decoding of QC LDPC codes," in *40th Annual Conf. on Info. Sciences and Syst.*, vol. 11, March . 2006, pp. 839–8446.

- [75] Z. Wang and Z. Cui, "Low-Complexity High-Speed Decoder Design for Quasi-Cyclic LDPC Codes," *IEEE Trans. VLSI Syst.*, vol. 15, pp. 104–114, 2007.
- [76] K. K. Gunnam, G. S. Choi, M. B. Yeary, and M. Atiquzzaman, "VLSI Architectures for Layered Decoding for Irregular LDPC Codes of WiMax," in *IEEE International Conference on Communications (ICC)*, June 2007, pp. 4542–4547.
- [77] X.-Y. Shih, C.-Z. Zhan, C.-H. Lin, and A.-Y. Wu, "An 8.29 mm² 52 mW Multi-Mode LDPC Decoder Design for Mobile WiMAX System in 0.13 μ m CMOS Process," *IEEE Journal of Solid-State Circuits*, vol. 43, pp. 672–683, Mar. 2008.
- [78] T. Mohsenin, D. Truong, and B. Baas, "Multi-Split-Row Threshold Decoding Implementations for LDPC Codes," in *IEEE International Symposium on Circuits and Systems (ISCAS'09)*, May 2009, pp. 2449–2452.
- [79] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic LDPC codes," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 985 – 994, 2009.
- [80] M. Mansour and N. Shanbhag, "A 640-Mb/s 2048-Bit Programmable LDPC Decoder Chip," *IEEE Journal of Solid-State Circuits*, vol. 41, pp. 684–698, March 2006.
- [81] M. Karkooti, P. Radosavljevic, and J. R. Cavallaro, "Configurable, High Throughput, Irregular LDPC Decoder Architecture: Tradeoff Analysis and Im-

- plementation,” in *IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 06)*, 2006.
- [82] F. G. Engineer, G. L. Nemhauser, and M. W. P. Savelsbergh, “Dynamic Programming-Based Column Generation on Time-Expanded Networks: Application to the Dial-a-Flight Problem,” *Journal on computing*, May 2010.
- [83] J. Anderson and S. Mohan, “Sequential Coding Algorithms: A Survey and Cost Analysis,” *IEEE Transactions on Communications*, vol. 32, pp. 169–176, 1984.
- [84] D. E. Knuth, *Art of Computer Programming*. Addison-Wesley Professional, 1998, vol. 3.
- [85] K. Amiri, J. R. Cavallaro, C. Dick, and R. M. Rao, “A High Throughput Configurable SDR Detector for Multi-user MIMO Wireless Systems,” *Journal of Signal Processing Systems (in press)*, 2010.
- [86] P. Radosavljevic, Y. Guo, and J. R. Cavallaro, “Probabilistically bounded soft sphere detection for MIMO-OFDM receivers: algorithm and system architecture,” *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 8, pp. 1318–1330, Oct. 2009.
- [87] M. Myllyla, M. Juntti, and J. R. Cavallaro, “Implementation aspects of list sphere decoder algorithms for MIMO-OFDM systems,” *ELSEVIER Signal Processing*, vol. 90, no. 10, pp. 2863–2876, Oct. 2010.

- [88] P. Luethi, C. Studer, S. Duetsch, E. Zraggen, H. Kaeslin, N. Felber, and W. Fichtner, "Gram-Schmidt-based QR decomposition for MIMO detection: VLSI implementation and comparison," in *IEEE Asia Pacific Conference on Circuits and Systems*, December 2008, pp. 830–833.
- [89] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithm operating in the log domain," in *IEEE Int. Conf. Commun. (ICC)*, 1995, pp. 1009–1013.
- [90] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Transactions on Communications*, vol. 44, pp. 1261 – 1271, Oct. 1996.
- [91] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Tran. on Information Theory*, vol. IT-20, pp. 284–287, Mar. 1974.
- [92] T. K. Blankenship, B. Classon, and V. Desai, "High-throughput turbo decoding techniques for 4G," in *Int. Conf. Third Generation Wireless and Beyond*, May 2002, pp. 137–142.
- [93] P. Salmela, R. Gu, S. Bhattacharyya, and J. Takala, "Efficient parallel memory organization for turbo decoders," in *Proc. European Signal Processing Conf.*, Sep. 2007, pp. 831–835.
- [94] "Multiplexing and channel coding, 3GPP TS 36.212 version 8.4.0, Sept. 2008."

- [95] O. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Trans. Inform. Theory*, vol. 52, pp. 1249–1253, Mar. 2006.
- [96] J. Sun and O. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. Inform. Theory*, vol. 51, Jan. 2005.
- [97] A. Nimbalkar, K. T. Blankenship, B. Classon, T. E. Fuja, and D. J. Costello, "Contention-free interleavers for high-throughput turbo decoding." *IEEE Transactions on Communications*, vol. 56, no. 8, pp. 1258–1267, Aug. 2008.
- [98] A. Nimbalkar, Y. W. Blankenship, B. K. Classon, and K. T. Blankenship, "ARP and QPP Interleavers for LTE Turbo Coding," in *IEEE Wireless Communications and Networking Conference*, April 2008, pp. 1032–1037.
- [99] P. Ampadu and K. Kornegay, "An efficient hardware interleaver for 3G turbo decoding," in *IEEE Radio and Wireless Conference*, Aug. 2003, pp. 199–201.
- [100] G. Masera, M. Mazza, G. Piccinini, F. Viglione, and M. Zamboni, "Low-cost IP-blocks for UMTS turbo decoders," in *27th European Solid-State Circuits Conference*, Sept. 2001, pp. 470–473.
- [101] S. S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," *International Journal of Satellite Communications*, vol. 16, pp. 23 – 46, Dec. 1998.

- [102] C. Schurgers, F. Catthoor, and M. Engels, "Memory optimization of MAP turbo decoder algorithms," *IEEE Trans. VLSI Syst.*, vol. 9, no. 2, pp. 305–312, April 2001.
- [103] S.-J. Lee, N. Shanbhag, and A. Singer, "Area-efficient high-throughput MAP decoder architectures," *IEEE Trans. VLSI Syst.*, vol. 13, pp. 921–933, Aug. 2005.
- [104] Y. Zhang and K. Parhi, "High-throughput radix-4 logMAP turbo decoder architecture," in *Asilomar Conf. on Signals, Syst. and Computers*, Oct. 2006, pp. 1711–1715.
- [105] C. Schurgers, F. Catthoor, and M. Engels, "Optimized MAP turbo decoder," in *IEEE Signal Processing Systems (SiPS)*, Oct. 2000, pp. 245–254.
- [106] R. Ratnayake, A. Kavcic, and G.-Y. Wei, "A High-Throughput Maximum a Posteriori Probability Detector," *IEEE Journal of Solid-State Circuits*, vol. 43, pp. 1846 – 1858, 2008.
- [107] A. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Sel. Areas Commun.*, vol. 16, pp. 260–264, Feb. 1998.
- [108] J. Dielissen and J. Huiskens, "State vector reduction for initialization of sliding windows MAP," in *2nd International Symposium on Turbo Codes and Related Topics*, Sept. 2000.

- [109] M. M. Mansour and N. R. Shanbhag, "VLSI architectures for SISO-APP decoders," *IEEE Tran. VLSI Syst.*, vol. 11, no. 4, pp. 627–650, Aug. 2003.
- [110] G. Maserà, G. Piccinini, M. Roch, and M. Zamboni, "VLSI architecture for turbo codes," in *IEEE Trans. VLSI Syst.*, vol. 7, 1999, pp. 369–3797.
- [111] Z. Wang, Z. Chi, and K. Parhi, "Area-efficient high-speed decoding schemes for turbo decoders," *IEEE Tran. on VLSI Syst.*, vol. vol.10, pp. 902–912, Dec 2002.
- [112] B. Bougard, A. Giulietti, V. Derudder, J.-W. Weijers, S. Dupont, L. Hollevoet, F. Catthoor, L. Van der Perre, H. De Man, and R. Lauwereins, "A scalable 8.7-nJ/bit 75.6-Mb/s parallel concatenated convolutional (turbo-) codec," in *IEEE International Solid-State Circuit Conference (ISSCC)*, Feb. 2003.
- [113] M. J. Thul, F. Gilbert, T. Vogt, G. Kreiselmaier, and N. Wehn, "A scalable system architecture for high-throughput turbo-decoders," *Journal of VLSI Signal Processing*, pp. 63–77, 2005.
- [114] G. Prescher, T. Gemmeke, and T. Noll, "A parametrizable low-power high-throughput turbo-decoder," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 5, Mar. 2005, pp. 25–28.
- [115] R. Dobkin, M. Peleg, and R. Ginosar, "Parallel interleaver design and VLSI architecture for low-latency MAP turbo decoders," *IEEE Trans. VLSI Syst.*, vol. 13, no. 4, pp. 427–438, 2005.

- [116] M. May, C. Neeb, and N. Wehn, "Evaluation of High Throughput Turbo-Decoder Architectures," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2007, pp. 2770–2773.
- [117] A. Tarable, L. Dinoi, and S. Benedetto, "Design of prunable interleavers for parallel turbo decoder architectures," *IEEE Comm. Lett.*, vol. 11, pp. 167–169, Feb. 2007.
- [118] R. Asghar, D. Wu, J. Eilert, and D. Liu, "Memory Conflict Analysis and Implementation of a Re-configurable Interleaver Architecture Supporting Unified Parallel Turbo Decoding," *Journal of VLSI Signal Processing*, July 2009.
- [119] T. Zhang, Z. Wang, and K. Parhi, "On finite precision implementation of low density parity check codes decoder," in *IEEE Int. Symposium on Circuits and Systems*, vol. 4, May 2001, pp. 202–205.
- [120] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Tran. on Information Theory*, vol. 42, no. 2, pp. 429 – 445, 1996.
- [121] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *IEEE GLOBECOM*, Oct. 2001, pp. 1036–1036.
- [122] C. Studer, N. Preyss, C. Roth, and A. Burg, "Configurable high-throughput decoder architecture for quasi-cyclic LDPC codes," in *IEEE Asilomar Conference*

- on Signals, Systems and Computers*, Oct 2008, pp. 1137–1142.
- [123] Z. Cui, Z. Wang, and Y. Liu, “High-Throughput Layered LDPC Decoding Architecture,” in *IEEE Transactions on Very Large Scale Integration Systems*, Apr. 2009, pp. 582–587.
 - [124] J. Hagenauer, E. Offer, and L. Papke, “Iterative decoding of binary block and convolutional codes,” *IEEE Tran. Info. Theory*, vol. 42, pp. 429–445, 1996.
 - [125] G. Masera, F. Quaglio, and F. Vacca, “Finite precision implementation of LDPC decoders,” in *IEEE Proc. Commun.*, vol. 152, Dec 2005, pp. 1098–1102.
 - [126] T. Zhang, Z. Wang, and K. Parhi, “On finite precision implementation of low density parity check codes decoder,” in *IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 4, May 2001, pp. 202–205.
 - [127] A. Abbasfar and K. Yao, “An efficient and practical architecture for high speed turbo decoders,” in *IEEE Vehicular Technology Conference*, vol. 1, 2003, pp. 337 – 341.
 - [128] “WARP : <https://www.warp.rice.edu/>.”
 - [129] K. Amiri, Y. Sun, P. Murphy, C. Hunter, J. Cavallaro, and A. Sabharwal, “Warp, a unified wireless network testbed for education and research,” in *IEEE International Conference on Microelectronic Systems Education*, San Diego, CA, June 2007, pp. 53 – 54.

- [130] X.-Y. Shih, C.-Z. Zhan, C.-H. Lin, and A.-Y. Wu, "A 19-mode 8.29mm² 52-mW LDPC Decoder Chip for IEEE 802.16e System," in *2007 Symposium on VLSI Circuits*, June 2007.
- [131] *Synfora PICO Extreme datasheet*, <http://www.synfora.com>, <http://www.synopsys.com/Systems/BlockDesign/HLS/Pages/default.aspx>.
- [132] V. Kathail, et al, "PICO: automatically designing custom computers," *Computer*, vol. 35, pp. 39 – 47, 2002.
- [133] *SpyGlass datasheet*, <http://www.atrenta.com>.
- [134] Y. Sun, G. Wang, and J. R. Cavallaro, "Multi-Layer Parallel Decoding Algorithm and VLSI Architecture for Quasi-Cyclic LDPC Codes," 2010, submitted to IEEE International Symposium on Circuits and Systems.
- [135] Z. He, P. Fortier, and S. Roy, "Highly-Parallel Decoding Architectures for Convolutional Turbo Codes." *IEEE Tran. VLSI Syst.*, vol. 14, no. 10, pp. 1147–1151, Oct. 2006.
- [136] M. Alles, T. Vogt, and N. Wehn, "FlexiChaP: A reconfigurable ASIP for convolutional, turbo, and LDPC code decoding," in *2008 5th International Symposium on Turbo Codes and Related Topics*, Sept. 2008, pp. 84–89.
- [137] M. Scarpellino, A. Singh, E. Boutillon, and G. Masera, "Reconfigurable Architecture for LDPC and Turbo Decoding: A NoC Case Study," in *IEEE 10th In-*

ternational Symposium on Spread Spectrum Techniques and Applications, Aug. 2008, pp. 671–676.

- [138] A. Niktash, H. Parizi, A. Kamalizad, and N. Bagherzadeh, “RECFEC: A Reconfigurable FEC Processor for Viterbi, Turbo, Reed-Solomon and LDPC Coding,” in *IEEE Wireless Communications and Networking Conference (WCNC)*, March 2008, pp. 605–610.